Graph partitions with proportional density and colouring constraints

Doctoral thesis

Clément Dallard

University of Portsmouth
School of Computing

The thesis is submitted in partial fulfilment of the requirements for the award of the degree of Doctor of Computer Science of the University of Portsmouth.

September 2019

Declaration

Whilst registered as a candidate for the above degree, I have not been registered for any other research award. The results and conclusions embodied in this thesis are the work of the named candidate and have not been submitted for any other academic award.

Acknowledgements

This three years journey, and obviously this resulting thesis, would not have been possible without the commitment of my supervisor Janka Chlebíková. You have always been thoughtful and generous. It has been a pleasure being your student!

I would also like to thank Cristina Bazgan with whom I had the chance to work with. Our collaboration was very instructive, and I wish we will have other opportunities to meet and work together again.

It has also been great to work with Thomas Pontoizeau. Who knows, we may finally be done with the cubic graphs one day.

When I first came to Portsmouth, I have collaborated with Niklas Paulsen, a funny man full of ideas. I wish you the best buddy, and hopefully we will meet again soon.

If there is someone in Portsmouth I have discussed and drunk coffee with, that must be my friend Ali. Thanks for your support, these three years would not have been the same without you.

Valentin, I have always greatly enjoyed our discussions and afternoon proofs! Thank you for offering your help and keeping me motivated.

Angie, you defied the English weather and Portsmouth's accent so we could be together. I am one lucky man.

Mamouchka, don't worry anymore, I have survived the English food. Thanks for your homemade tapenade supply.

Baboun, thank you for your encouragements and moral support. You have been more helpful than you can realise.

Benouille, come hug your brother!

Abstract

Motivated by real life challenges and pure research curiosity, this thesis considers several combinatorial problems related to combinatorial optimisation, computational complexity and graph theory. In each chapter, our goal is to refine our understanding of the complexity of the studied problem. When considering NP-hard problems, we eventually circumvent the expected exponential complexity by providing parameterized and approximation algorithms. We also study the problems on restricted type of instances and propose efficient polynomial-time algorithms.

We study the notions of *proportional density* and define a *proportionally dense subgraph* as a subgraph whose vertices have proportionally as many neighbours inside the subgraph as in the whole graph. This notion combines local and global properties of the subgraph, an interesting paradigm rarely encountered in graph theory. Two problems related to proportionally dense subgraphs are studied, from the perspectives of structural graph theory, computational complexity and approximation. Then, we focus on a graph partitioning problem on vertex-coloured graphs, and study its complexity on restricted classes of caterpillars with bounded hair-length and planar graphs. Our contributions expose a gap in the complexity of the problem with regard to the hair-length and the maximum degree of the graph. Lastly, we propose a complexity study of a scheduling problem with fixed route and soft time constraints. We show that the problem's complexity inherently depends on the ride time constraints and give several polynomial-time algorithms for restricted type of instances.

Contents

Chapter 1

Preliminaries

Outline

The goal of this chapter is to introduce the basic concepts of graph theory and computational complexity theory that are needed to apprehend the next chapters. We will cover the notions of decision and optimisation problems, complexity classes and approximation.

This chapter aims to provide the basic definitions of the concepts needed later in this thesis, such as the complexity classes P and NP, polynomial-time reductions, approximation algorithms. First, we define some graph notations that we use throughout the thesis. Then, we discuss the computational complexity of decision and optimisation problems. Finally, we give a short overview of the content of the next chapters.

1.1. Graphs: notations and definitions

An *undirected graph* is a pair $G = (V, E)$, where V is a set of *vertices* and E is a set of *edges*, which are elements of $V \times V$. The vertices u and v of an edge $e = \{u, v\}$ are called the *endpoints* of e, and u and v are said to be *adjacent*. We also say that an edge f is *incident* to a vertex w if $w \in f$, and f is *incident* to another edge f' if $f \cap f' \neq \emptyset$.

A *loop* is an edge $\{u, u\}$, for some vertex $u \in V$. When the edges are ordered pairs of vertices, then the graph is *directed* and the edges are usually called *arcs* to avoid any ambiguity.

In this thesis, we only consider *finite* graphs, that is, graphs with a finite number of vertices. Also, unless stated otherwise, all graphs are unweighted, undirected and *simple*, *i.e.* without loops or multiple edges.

A *path* is a sequence of distinct edges which joins a sequence of distinct vertices. For simplicity, depending on the context, we may define a path as a sequence of edges or as a sequence of vertices. A *cycle* is a sequence of distinct edges which joins a sequence of vertices in which the only repeated vertices are the first and last vertices. Again, we may define a cycle as a sequence of edges or as a sequence of vertices.

A *subgraph* $G' = (V', E')$ of G is a graph such that $V' \subseteq V$, $E' \subseteq E$ and each edge in E' has both endpoints in V'. A subgraph is *connected* if there exists a path between any two vertices in the subgraph. A *connected component* is a maximal connected subgraph. A subgraph $G' = (V', E')$ is an *induced subgraph* of G if E' contains all the edges of E that have both endpoints in V'. For a given $S \subseteq V$, $G[S]$ denotes the induced subgraph of S in G. The set S is a *clique* if $G[S]$ contains all possible edges between the vertices in S, and it is an *independent set* if $G[S]$ does not contain any edge. Let $u \in V$ be a vertex in G, then $N(u) := \{v \in V : \{u, v\} \in E\}$ is the *neighbourhood* of u, $N[u] := N(u) \cup \{u\}$ is the *closed neighbourhood* of u and $d(u) := |N(u)|$ is the *degree* of u. We write $d_S(u) := |N(u) \cap S|$ for the degree of u in $G[S]$. We denote by $\bar{S}$ the *complement*

of S in G, that is, $\bar{S} := V \setminus S$. The *cut* $(S, \bar{S})$ is a partition of V into two subsets S and $\bar{S}$, the *cut-set* is the set of edges with endpoints in different subsets, and the size of the cut is the number of edges in the cut-set.

Besides these notations, we also consider some common classes of graphs. A *complete graph* is a graph that contains all possible edges between its vertices. A *planar graph* is a graph that can be drawn in the plane such that the edges only intersect at their endpoints. A *bipartite graph* is a graph whose vertex-set can be partitioned into two parts such that two adjacent vertices are in different parts. A *complete bipartite graph* is a bipartite graph whose edge-set contains every possible edge that connects vertices in different parts. A *star* is a complete bipartite graph with one part of size 1; A *tree* is a connected and acyclic graph (a tree is bipartite). A *caterpillar* is a tree which contains a central path such that all vertices are within distance 1 of the path, called the *backbone*. By extension, a *k-caterpillar* is a tree such that all vertices are within distance k of the backbone.

We recommend the reading of [15, 87] for the definitions of other commonly used graph notations and terminology. Other notations, terms and graph classes may also be introduced later in this thesis.

1.2. Computational complexity

This section will provide some basic definitions in computational complexity. We made the choice to keep these definitions as simple as possible and refer to the cited literature for formal definitions using languages and Turing machines.

1.2.1. Algorithms

An algorithm is a step-by-step procedure that receives an input and performs some computation on it. We can assume that an algorithm always returns an output, even if constant. Algorithms fall into two categories, *deterministic* and *nondeterministic*.

Definition 1.1: Deterministic and nondeterministic algorithms

- A *deterministic algorithm* will always go through the same sequence of states and return the same output for a fixed given input.
- A *nondeterministic algorithm* may go through a different sequence of states and return different outputs for a fixed given input.

A nondeterministic algorithm can be seen as an algorithm that can visit all possible computational paths. This means that, instead of branching at each alternative like a deterministic algorithm would, every alternative can run simultaneously [45].

The *computational complexity* of an algorithm is the amount of resources needed to run the algorithm. This amount is expressed as a function of the size of the input. For instance, an algorithm with complexity $f(n)$ means that the algorithm needs $f(n)$ resources to solve any instance of size n. If not clearly specified, the resource is always the time, and the computational complexity of an algorithm is called the *running time* (of the algorithm). The running time is generally expressed as the number of elementary operations which are assumed to take constant time on a computer. Hence, the running time of an algorithm can be expressed as the number of elementary operations multiplied by a constant factor, which depends on the computer. The space complexity of an algorithm can also be studied but is not considered in this thesis[1].

In this thesis we express the complexity of an algorithm with the O *notation*.

Definition 1.2: O notation

Given two functions f and g, we say that $f = O(g)$ ("f is big Oh of g") if there exist two positive constants α and x_0 such that, for all $x \geq x_0$, $f(x) \leq \alpha \cdot g(x)$.

Other asymptotic notations can be encountered in the literature, such as the o *notation* (f is dominated by g asymptotically), the Θ *notation* (f is bounded both above and below by g asymptotically) and the Ω *notation* (f is bounded below by g asymptotically).

1.2.2. Decision problems, P and NP

A *decision problem* is a problem that can be posed as a yes-no question of the input values. We are particularly interested in the problems for which a *certificate* (also called *proof* or *witness* in the literature) can be verified in polynomial time. Informally, if someone gives us a "solution" for some instance of a problem, we want to be able to check whether the "solution" is valid in efficient time.

[1]Since we consider decision problems in NP and NP $\subseteq$ PSPACE, the problems we study can be solved using a polynomial amount of space.

Definition 1.3: NP class

The class NP is the set of all decision problems for which the problem instances, where the answer is "yes", have certificates verifiable in polynomial time.

An equivalent definition of the class NP is the set of decision problems for which there exists a nondeterministic polynomial-time algorithm solving the problem [4].

The *complement* of a decision problem is the decision problem resulting from reversing the "yes" and "no" answers, and the class coNP contains the decision problems whose complements are in NP.

Some problems in NP are known to be polynomial-time solvable, and they define a subset of NP.

Definition 1.4: P class

The class P is the set of all decision problems that can be solved in polynomial time with a deterministic algorithm.

While some problems in NP are proved to belong to P, others are not, despite decades of research. There lies the P versus NP problem : There seems to exist a dichotomy of the problems in NP between problems that are "easy to solve" (in P) and problems that are "hard to solve". As described by Cook in [28], the P versus NP problem is "*to determine whether every language accepted by some nondeterministic algorithm in polynomial time is also accepted by some (deterministic) algorithm in polynomial time.*" It is one of the seven Millennium Prize Problems[2].

To describe this complexity dichotomy in further details, we need to define the concept of *polynomial-time reduction*.

Definition 1.5: Polynomial-time reduction

Let A and B be two decision problems. A *polynomial-time reduction* is a function f that maps instances of B into instances of A in polynomial-time such that, for any instance x of B, x is a yes-instance if and only if $f(x)$ is a yes-instance. We say that B (polynomial-time) *reduces* to A.

We can now formally define the notions of *NP-hardness* and *NP-completeness*.

[2] https://www.claymath.org/millennium-problems/p-vs-np-problem

Definition 1.6: NP-hardness and NP-completeness

A decision problem A is NP-hard if every problem B in NP can be reduced in polynomial time to A. Furthermore, if A is NP-hard and belongs to NP, then A is NP-complete.

Informally, an NP-hard problem is at least as hard as any problem in NP. A polynomial-time reduction is one of the most useful tools at the disposal of the researcher for characterising the complexity of a problem. Its main use is to prove that a problem A is NP-hard by showing that there exists an NP-hard problem B and a polynomial-time reduction from B to A. But a polynomial-time reduction can also prove that a problem is in P: If a problem A can be reduced to a problem B in P, then A is obviously in P as well.

Even though not discussed in this, it is interesting to mention that, under the assumption that $\mathsf{P} \neq \mathsf{NP}$, there exist problems within NP that are not in P and not NP-complete [62] (these problems are called NP-intermediate).

Cook proved that the Boolean Satisfiability problem, commonly known as SAT, is NP-hard by showing that any problem in NP can be reduced to SAT [29]. An instance is a formula in conjunctive normal form (CNF), also called a *CNF formula*, which is a conjunction of one or more clauses, where a clause is a disjunction of literals. When each clause contains at most ℓ variables, we say that the formula is a *ℓ-CNF formula*. In the same paper, the 3-SAT problem where each clause contains *exactly* 3 variables has also been proved NP-hard. If it is possible to assign $True$ or $False$ values to all variables such that the formula evaluates to $True$, then the formula is *satisfiable* and the assignment is *satisfying*. On the other hand, if there is no satisfying assignment of the formula, then the formula is *unsatisfiable*.

SAT

Input: A CNF formula ϕ.
Question: Is there a satisfying assignment of ϕ?

Note that the problems SAT and 3-SAT also belongs to NP, and therefore are NP-complete.

Let us illustrate the concept of polynomial-time reduction by reducing 3-SAT to the Min Vertex Cover problem, and thus proving that the latter is NP-hard. A *vertex cover* is a subset of vertices S such that each edge contains at least one endpoint in S. This proof comes from [47] and is the first known result showing

that MIN VERTEX COVER is NP-hard. We safely assume that a clause does not contain the positive and negative literals of a same variable, otherwise we can simply remove the clause from the formula.

Min Vertex Cover

Input: A graph G and an integer k.
Question: Is there a vertex cover of size at most k in G?

The general idea of Construction 1.1 is to create a graph from a 3-CNF formula where each variable is represented by a *variable gadget* and each clause is represented by a *clause gadget*.

Construction 1.1

Let ϕ be a 3-CNF formula with exactly 3 variables per clause, that is, a set of m clauses $C_1, C_2, \ldots, C_m$ on n variables $x_1, x_2, \ldots, x_n$. We construct the graph $G = (V, E)$ as follows:

- for each variable x_i, create a *variable gadget* made of two adjacent vertices v_i and $\bar{v}_i$;
- for each clause C_j, create a *clause gadget* T_j, which is a clique of size 3 (a triangle);
- finally, for each variable x_i of a clause C_j, connect one vertex in T_j to v_i if the literal $x_i \in C_j$ or to $\bar{v}_i$ if $\bar{x}_i \in C_j$, and such that the vertices in T_j have degree 3 (connected to exactly one vertex outside the clause gadget).

Of course, Construction 1.1 can be done in polynomial time (see Fig. 1.1 for an example).

Theorem 1.1

MIN VERTEX COVER is NP-complete.

Proof. Of course, MIN VERTEX COVER is in NP. Let ϕ be a 3-CNF formula with exactly 3 variables per clause, that is, a set of m clauses $C_1, C_2, \ldots, C_m$ on n variables $x_1, x_2, \ldots, x_n$, and consider the graph $G = (V, E)$ obtained through Construction 1.1. We claim that ϕ is satisfiable if and only if there exists a vertex cover of size at most $n + 2m$ in G.

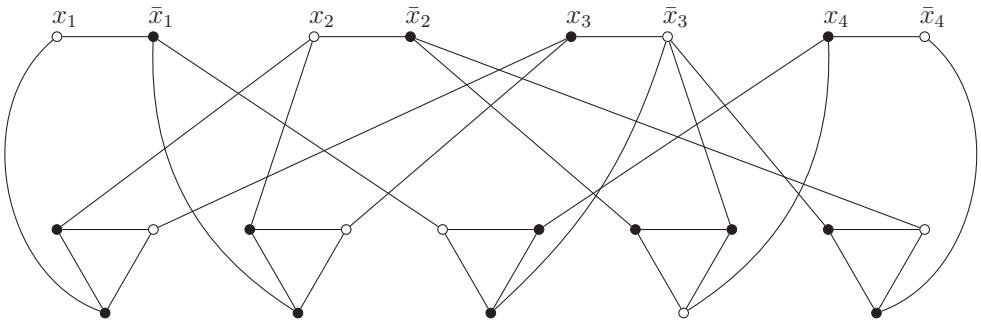


Figure 1.1: Example of a graph obtained through Construction 1.1 with the 3-CNF formula $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4)$. Vertices in black belong to a vertex cover of size $n + 2m = 14$, and hence $x_1 := False$, $x_2 := False$, $x_3 := True$ and $x_4 := True$.

Let β be a satisfying assignment of ϕ and S be an empty set of vertices. For each variable x_i, if x_i is set to $True$ in β, then add the vertex v_i to S, otherwise add the vertex $\bar{v}_i$ to S. At this point, $|S| = n$ and S covers all the edges between the vertices representing the literals of the variables. Now, consider a clause gadget T_j representing a clause C_j and note that there exists $u_j \in T_j$ adjacent to a vertex in S, since β satisfies ϕ. Then, add the vertices in $T_j \setminus \{u_j\}$ to S. Note that all the edges of the clause gadget and between the clause gadget and the corresponding variable gadgets are covered by S. Once all the clause gadgets have been processed, S is a vertex cover of size $n + 2m$.

Let S be a vertex cover in G of size at most $n + 2m$. It is easy to see that S contains at least two vertices for each clause gadget and at least one vertex for each variable gadget. Therefore, S has size exactly $n + 2m$ and contains exactly two vertices per clause gadget and one vertex per variable gadget. We set a variable x_i to $True$ in β if the vertex v_i belongs to S, otherwise we set it to $False$. Note that doing so, each variable is assigned one (and only one) value because S contains exactly one vertex per variable gadget. Now, suppose that there exists a clause C_j which is not satisfied by β and let u_j be the only vertex in T_j which does not belong to S. Without loss of generality, let v_i be the vertex from a variable gadget adjacent to u_j. Since S is a vertex cover and $u_j \notin S$, then $v_i \in S$, which implies that $x_i = True$ in β. A contradiction with the fact that the clause is not satisfied. ■

In Chapters 3 to 5 we make use of polynomial-time reductions to show that the problems Max PDS, Colourful Components and Colourful Partition are NP-hard on restricted classes of graphs, and the problem Min Pickup-Delivery Scheduling is NP-hard when ride time constraints are soft.

1.2.3. Optimisation problems, PO and NPO

In this subsection we are interested in *optimisation problems* where the goal is to find an *optimal solution* for a given input.

Definition 1.7: Optimisation problem

An *optimisation problem* is a quadruple (I, f, c, g) such that:

- I is a set of instances,
- given an instance $x \in I$, $f(x)$ is the set of feasible solutions,
- given $x \in I$ and $y \in f(x)$, $c(x, y)$ denotes the cost of y and $c(x, y)$ is computable in polynomial time, and
- g is a goal, either *maximisation* or *minimisation*.

An optimal solution is, therefore, a feasible solution with maximum or minimum cost (depending on the goal of the problem).

Note that an optimisation problem is always associated with a decision problem asking whether there exists a solution with cost *at least* k (if the goal is to maximise the cost) or *at most* k (if the goal is to minimise the cost). We therefore talk about the *decision variant* and *optimisation variant* of a problem.

Definition 1.8: NPO class

The class NPO contains all optimisation problems whose associated decision problem is in NP.

Definition 1.9: PO class

The class PO contains all optimisation problems that can be solved in polynomial time with a deterministic algorithm.

Of course, the inclusion PO $\subseteq$ NPO holds. Note that, given an optimisation problem A, if its decision variant is NP-hard, then necessarily its optimisation variant is NP-hard. We refer the reader to [8] for more details.

The notion of *NPO-completeness* also exists, and says that a problem A is NPO-complete if it belongs to NPO and there exists a PTAS-reduction from every problem in NPO. We do not cover the definition of PTAS-reduction here but we redirect the reader to [31].

1.2.4. Dealing with the complexity

The *Exponential Time Hypothesis* states that 3-SAT cannot be solved in subexponential time in the worst case. Note that some NP-complete problems can be solved in subexponential time (but not polynomial), for example, the Max Independent Set can be solved in subexponential time on planar graphs. However, when an instance of 3-SAT of size n is reduced to an instance of Max Independent Set on a planar graph, the latter graph contains $\Theta(n^2)$ vertices, so an exponential lower bound of $O(2^{cn})$ for 3-SAT translates into a lower bound of $O(2^{\sqrt{cn}})$.

In order to comprehend the computational complexity of an NP-hard problem, two complementary methods are available. On the one hand, one may decrease the values of the exponent and the base in the function expressing the computational complexity of the algorithm (this often implies the design of complex algorithms and fine complexity analysis). On the other hand, one can prove complexity lower bounds of the problem. For a short paper on the subject, see [76].

While the above methods are interesting to classify problems, the complexity of the problems is not polynomial in the size of the input, and therefore quickly intractable even for relatively small instances. In this section, we discuss three different approaches: restricting the problem on special types of instances, approximating the solution, and designing parameterized algorithms and kernels.

1.2.4.1. Restriction of the problem on special types of instances

Many structural properties of an instance can be restricted, thus allowing us to design efficient algorithms. In the general case, it is sometimes possible to design polynomial-time algorithms for restricted types of instances.

Taking the example of Min Vertex Cover again, it is not difficult to prove that the problem is polynomial-time solvable on bipartite graphs. This result is known as *König's theorem*, which in fact proves that in any bipartite graph the size of a maximum matching (a set of pairwise non-incident edges of maximum size) equals the size of a minimum vertex cover (see [15] for a proof).

For graph problems, many structural properties can be restricted, thus allowing us to design efficient algorithms. Also, one can focus on minor-free classes of graphs [66], for instance, planar graphs forbid $K_{3,3}$ (complete bipartite graphs with two parts of size 3) and K_5 (the complete graph on 5 vertices) as minors. A less restrictive but common technique is to forbid (induced) subgraphs. For example, Min Vertex Cover is polynomial-time solvable on P_5-free graphs [65] (graphs with no induced paths on 5 vertices), even though NP-hard on cubic graphs (all the vertices have degree 3) [2].

Of course, this restriction can be done together with the other techniques presented in the next subsections to obtain even more efficient algorithms.

1.2.4.2. Approximation and inapproximability

Assuming that P $\neq$ NP, the optimality of a solution for an NP-hard problem comes at a price: an exponential running-time. However, it is sometimes interesting to obtain a "good" but not necessarily optimal solution in a shorter time. This is exactly what approximation algorithms intend to do, trading the optimality of a solution for a gain of time. We refer the reader to [8, 84] for detailed definitions and advanced notions of approximation.

Let A be an optimisation problem and x be an instance of A. We denote by $OPT_A(x)$, or simply $OPT(x)$ if clear from context, the value of an optimal solution for x, and by $|x|$ the size of the instance.

Definition 1.10: Approximation ratio

Given an an optimisation problem A and an instance x of A, the *approximation ratio*, also called *performance ratio*, of a solution y to an instance x of A is defined as

$$R_A(x,y) = \max\left\{\frac{c_A(x,y)}{OPT(x)}, \frac{OPT(x)}{c_A(x,y)}\right\}.$$

A *polynomial-time approximation algorithm* is a generic name to describe an algorithm that runs in polynomial time and returns approximated solutions with a *provable guarantee* on the ratio between the optimal and the returned solution.

Definition 1.11: Approximation algorithm

Given an an optimisation problem A and an instance x of A, an *approximation algorithm* for A is an algorithm that returns a solution y to x such that $R_A(x,y) \leq f(|x|)$ for some function f.

Depending on the type of the function f, we can classify optimisation problems into complexity classes depending on difficulty to find an approximation algorithm for them. Let us first define the class APX that we will also encounter later in this thesis.

Definition 1.12: APX class

The class APX is the class of optimisation problems in NPO that admit a polynomial-time approximation algorithm whose approximation ratio is bounded by a constant.

A problem in APX with ratio bounded by a constant α is called a *polynomial-time α-approximation algorithm*, or more generally a *constant-factor polynomial-time algorithm*. Problems having an approximation algorithm with approximation ratio bounded by $\alpha \cdot log(|x|)$ form the class $log - \mathsf{APX}$. In the same way, problems admitting an approximation algorithm with approximation ratio bounded by $\alpha \cdot p(|x|)$, with p a polynomial, form the class $poly - \mathsf{APX}$.

To give an example of a problem in APX, we consider the optimisation variant of Min Vertex Cover and prove that there exists a polynomial-time 2-approximation algorithm for the problem.

Min Vertex Cover

Input: A graph G.
Output: A vertex cover in G of minimum size.

Theorem 1.2

Min Vertex Cover is polynomial-time 2-approximable.

Proof. Let $G = (V, E)$ be a graph, $G' = (V', E')$ be a copy of G and S be an empty set of vertices. Find a maximal matching M in G, that is, a maximal set of edges pairwise non-incident. Note that a maximal matching can be otained greedily in $O(|E|)$. For each edge $\{u, v\}$ in M, add u and v to S. Obviously, because M is maximal, then S is a vertex cover. Moreover, if $\{u, v\} \in M$, then every vertex cover contains u or v, otherwise the edge is not covered. Therefore, any vertex cover contains at least $|M|$ vertices, that is, $OPT(G) \geq |M|$. This means that $OPT(G) \leq |S| \leq 2 \cdot OPT(G)$, and therefore Min Vertex Cover is polynomial-time 2-approximable. ■

Corollary 1.2.1

MIN VERTEX COVER is in APX.

In Section 3.4, we give a polynomial-time approximation algorithm proving that MAX PDS can be approximated within an approximation of 2, hence showing that the problem belongs to APX.

Note that, for some problems in NPO, it is possible to arbitrarily fix the approximation ratio to any constant. Suppose that A is a minimisation problem, then a *polynomial-time approximation scheme* (PTAS) is an approximation algorithm that, given an instance x, returns a solution y such that $c_A(x, y) \leq (1 + \varepsilon) \cdot OPT(x)$ for any given $\varepsilon > 0$. If A is a maximisation problem, then $c_A(x, y) \geq (1 - \varepsilon) \cdot OPT(x)$. Note that its running time can be exponential in $\frac{1}{\varepsilon}$. The optimisation problems in NPO that admit a polynomial-time approximation scheme form the class PTAS, and if the running time is polynomial in the size of the instance and in $\frac{1}{\varepsilon}$, they form the class FPTAS.

We naturally obtain the following inclusions:

$$\mathsf{PO} \subseteq \mathsf{FPTAS} \subseteq \mathsf{PTAS} \subseteq \mathsf{APX} \subseteq \mathit{log} - \mathsf{APX} \subseteq \mathit{poly} - \mathsf{APX} \subseteq \mathsf{NPO}\,.$$

It is also interesting to prove lower bounds on the approximation ratio of a problem, or more generally to prove that a given problem does or does not belong to some approximation class, (*e.g.* PTAS or APX). In this thesis, we use two main techniques for such proofs: *approximation-preserving reductions* and *gap reductions*.

Definition 1.13: Approximation-preserving reduction

An *approximation-preserving reduction* between two optimisation problems A and B is a pair of functions (f, g) such that:

- f maps instances x of A to instances x' of B,
- g maps solutions y' of B to solutions y of A, and
- g preserves some guarantee on the approximation ratio of the output solutions.

This means that if A reduces to B via an approximation-preserving reduction, then given an instance x of A and an (approximation) algorithm for B, it is possible to convert the instance x of A into an instance x' of B, run the algorithm on x', and recover a solution for A that has some guarantee on the approximation ratio.

Some approximation-preserving reductions also preserve the membership in a complexity class C: If A reduces to B with such an approximation-preserving reduction and B belongs to C, then A belongs to C. Note that if a problem A is C-hard and A reduces to B through an approximation-preserving reduction, then B is C-hard [31].

In Section 3.3.1 and Section 5.3.1, we use an *L-reduction* to prove the APX-hardness of two different problems.

Definition 1.14: L-reduction

Let A and B be two optimisation problems. An *L-reduction* from A to B is an approximation-preserving reduction (f, g) such that for any instance x of A and any solution y' of $f(x)$:

- $OPT_B(f(x)) \leq \alpha \cdot OPT_A(x)$, with α a constant, and
- $|OPT_A(x) - c_A(x, g(y'))| \leq \beta \cdot |OPT_B(f(x)) - c_B(f(x), y')|$, with β a constant.

We now introduce the concept of *gap reductions*, a powerful technique to prove hardness results. Gap reductions are usually used to prove lower bounds on the best possible approximation ratio for a problem (or eventually the inapproximability of the problem).

Definition 1.15: Gap introducing reduction

A *gap-introducing reduction*, also called *gap-producing reduction*, from a decision problem A to a minimisation problem B consists of functions f and β along with a polynomial-time algorithm which given an instance x of A outputs an instance x' of B such that:

- if x is a yes-instance, then $OPT_B(x') \leq f(x')$, and
- if x is a no-instance, then $OPT_B(x') > \beta(|x'|) \cdot f(x')$.

If B is a maximisation problem, then:

- If x is a yes-instance, then $OPT_B(x') \geq f(x')$.

- If x is a no-instance, then $OPT_B(x') < \frac{f(x')}{\beta(|x'|)}$.

The gap, $\beta(|x|)$ is the *hardness factor* of the gap-introducing reduction. This means that a polynomial-time approximation algorithm for B with approximation ratio $\beta(|x|)$, or better, would be able to distinguish between the two cases, and ultimately solve problem B in polynomial time. Therefore, if A is NP-hard, then it is NP-hard to approximate B within $\beta(|x|)$.

It is now possible to use this result and obtain a *gap-preserving* reduction between two optimisation problems.

Definition 1.16: Gap-preserving reduction

Let A and B be optimisation problems. A *gap-preserving* reduction from A to B comes with four functions f_A, f_B, α and β. Assume that A and B are minimisation problems. Then, given an instance x of A, the reduction returns in polynomial time an instance x' of B such that:

- $OPT(x) \leq f_A(x) \implies OPT(x') \leq f_B(x')$;
- $OPT(x) \geq \alpha(|x|) \cdot f_A(x) \implies OPT(x') \geq \beta(|x'|) \cdot f_B(x')$.

If A and B are maximisation problems, then:

- $OPT(x) \geq f_A(x) \implies OPT(x') \geq f_B(x')$;
- $OPT(x) \leq \frac{f_A(x)}{\alpha(|x|)} \implies OPT(x') \leq \frac{f_B(x')}{\beta(|x'|)}$.

Approximating B in polynomial-time within a ratio of $\beta(|x'|)$ would imply that we can distinguish between the two cases for A. Hence, if it is NP-hard to approximate A within $\alpha(|x|)$, then it must be NP-hard to approximate B within $\beta(|x'|)$.

A similar technique has been used in [35] to prove that Min Vertex Cover cannot be approximated within 1.3606 unless P = NP. If the *unique games conjecture* (UGC) [58] is true, then it can be showed that Min Vertex Cover cannot be approximated within a constant factor strictly smaller than 2 [59]. Another way to state these results is to say that any polynomial-time approximation algorithm that solves Min Vertex Cover has an approximation ratio greater than 1.3606 unless P = NP and greater than $2 - \varepsilon$, for any $\varepsilon > 0$, unless UGC fails. Hence, Min Vertex Cover is APX-hard, and since we proved that Min Vertex Cover is in APX, we obtain the following corollary.

Corollary 1.2.2

MIN VERTEX COVER is APX-complete.

In Section 3.3, we use a gap-preserving reduction to give a constant lower bound on the approximation ratio of any polynomial-time algorithm for the MAX PDS problem and therefore prove that the problem is APX-hard.

1.2.4.3. Parameterized complexity

To effectively tackle an NP-hard problem and understand which parts of the instance influence the complexity, one can study the *parameterized complexity* of the problem [32, 38, 46]. The idea is to express the complexity of a decision problem by a parameter k. The natural parameter is usually the size of the solution, but many other parameters can be compared, such as treewidth, clique number, independent number, chromatic number...

A simple brute force algorithm usually implies a complexity of type $O(c^{O(n)})$, for c a constant and n the size of the instance. The goal here is to avoid the appearance of n in the exponent.

Definition 1.17: XP class

The class XP contains the decision problems that can be solved in time $O(n^{f(k)})$, for some computable function f.

While the complexity of a problem in XP can be interesting when $f(k)$ is small, a major improvement would be to make sure that n does not participate in the exponent or the base of the exponential.

Definition 1.18: FPT class

The class FPT contains the decision problems for which an algorithm with complexity $O(f(k) \cdot n^c)$ exists, for some computable function f.

An algorithm with this kind of complexity if called a *fixed-parameter algorithm* (also known as FPT algorithm). Often, such complexities are written $O^*(f(k))$, which drops the polynomial factor in n. Problems in FPT are said to be *fixed-parameter tractable*.

An alternative to finding such an algorithm is to design a *kernel* for the problem. A kernel is an instance (I', k') such that:

- the size of I' must be bounded by a function of k',
- k' must be bounded by a function of k,

- (I', k') is obtained in $O(f(k) \cdot n^c)$ time, and
- there is a solution to (I', k') if and only if there is a solution to (I, k).

It can be shown that a problem is FPT if and only if it admits a kernel. Thus, kernelization is another way of defining fixed-parameter tractability. For example, the Min Vertex Cover problem admits a kernel of size at most $3k$ [26], with k the natural parameter of the problem, and hence can be solved in time $O^*(2^{3k})$.

Unfortunately, not all problems are fixed parameter tractable, and such a complexity cannot be obtained. We refer the reader to [37] for a definition of the W-hierarchy and its implications on the (parameterized) complexity of problems.

1.3. Overview of the thesis

In this thesis, we study several different problems related to graph partitioning, computational complexity and combinatorial optimisation.

In Chapter 2, we consider the problem of partitioning a graph into two parts such that each part induces a *proportionally dense subgraph* (PDS), namely a 2-*PDS partition*. A PDS is a subgraph in which each vertex has *proportionally* as many neighbours within the subgraph as in the whole graph. We say that a PDS is a subgraph that respects the notion of *proportional density*. We provide the first known examples of graphs that do not admit a 2-PDS partition. In fact, we are able to construct an infinite family of graphs without a 2-PDS partition. The existence of such graphs was left as an open question in [10]. Then, we provide another class of graphs which have a 2-PDS partition but where one of the parts necessarily induces a disconnected subgraph.

Inspired by the notion of proportional density and PDS, in Chapter 3 we consider the problem of finding a PDS of maximum size, namely the Max PDS problem. We prove several hardness results on the problem, such as the APX-hardness on split graphs and the NP-hardness on bipartite graphs. We also show that deciding if a PDS is inclusion-wise maximal is coNP-hard, even on bipartite graphs. Nevertheless, we give a polynomial-time $(2-\frac{2}{\Delta+1})$-approximation algorithm for the problem, where Δ is the maximum degree of the graph. This, in turn, proves that the problem is APX-complete. Then, we consider the problem on Hamiltonian cubic graphs and show that all such graphs except two have a PDS of size $\lfloor \frac{2n+1}{3} \rfloor$, which we prove to be an upper bound on the size of a PDS in cubic graphs. Such a PDS is connected and can be found in linear time if a Hamiltonian cycle is given.

In Chapter 4, we study two graph partition problems on vertex coloured graphs. The goal is to partition the graph into *colourful components* which are connected components with no two vertices of the same colour while minimising the number of components (Colourful Partition) or minimising the number of edges with endpoints in different components (Colourful Components). These problems, originally motivated by comparative genomics, have been studied in several classes of graphs but questions regarding their complexity were left open. We first prove that both problems are NP-hard on quaternary 2-caterpillars, ternary 3-caterpillars and binary 4-caterpillars, answering at the same time an open question regarding their complexity on trees with maximum degree at most 5 [18]. Nonetheless, we show that the problems are linear-time solvable on 1-caterpillars, without any restriction on the degrees and even if the backbone induces a cycle. Our algorithm outperforms the previously best known quadratic algorithm for paths. Finally, we answer an open question regarding the complexity of Colourful Components in ℓ-coloured graphs [18] with maximum degree at most 5 by proving that the problem remains NP-hard on 5-coloured planar graphs with maximum degree 4 and on 12-coloured planar graphs with maximum degree 3.

In Chapter 5, we give several contributions to a well-studied scheduling problem in the context of Dial-A-Ride problems. An instance of our problem is a fixed route of stops, where each stop is either a pickup or a delivery of a request. Each stop is associated with a *time window constraint* and a corresponding penalty function when the former is violated. Also, each request is associated with a *ride time constraint*, which is an upper bound on the time between the scheduled pickup and the scheduled delivery, and a corresponding penalty function. Our first contribution is to show that this problem is NP-hard if and only if the ride time constraints are soft, *i.e.* if such constraints can be violated. Then, we consider several subclasses of instances where the problem becomes polynomial, for instance when the value of the maximum ride time is bounded by a constant and when the stops in the sequence are ordered such that all pickups appear before all the deliveries.

Finally, in Chapter 6, we review the problems studied in this thesis and summarise our contributions.

Chapter 2

Partition into Two Proportionally Dense Subgraphs

Outline

In this chapter, we are interested in a problem of partition of a graph into exactly two *proportionally dense subgraphs* (PDS), namely a 2-*PDS partition*. In a PDS, each vertex must have proportionally as many neighbours in its PDS than in the other. We answer in the negative to a question left open in [10] regarding the decidability of the problem: *Do all graphs admit a* 2-*PDS partition?* Then, we investigate the cases where a 2-PDS partition exists but at least one of the PDS's is disconnected.

Some of the results presented in this chapter appear in the following paper:

- C. Bazgan, J. Chlebíková and C. Dallard, 'Graphs without a partition into two proportionally dense subgraphs', *arXiv e-prints*, Submitted to *Information Processing Letters*, 2018. arXiv: `1806.10963 [cs.DM]`.

A journal version is currently under review for publication in a selected journal.

2.1. Introduction

In this chapter, we are interested in the graphs that cannot be partitioned into two (connected) subgraphs respecting the notion of *proportional density*. First, let us mention some NP-hard problems that consider a partition into two parts, that are close to our notion of 2-*PDS partition*.

The Satisfactory Partition problem, introduced by Gerber and Kobler [50], asks whether a graph can be partitioned into two parts such that every vertex is adjacent to more vertices in its own part than in the other. The problem has been extensively studied, both from complexity and approximation perspectives [13, 49, 50, 80].

The Bisection problem asks whether there exists a partition into two parts of the same size such that the cut between the two parts is smaller or equal than a given integer k. However, several polynomial-time approximation algorithms exist [41, 77] and the problem is FPT [33]. An interesting work from Wagner and Wagner investigates the complexity of intermediate problems between Min Cut and Bisection [86], *i.e.* problems deciding if there exists a partition into two parts of bounded sizes whose cut is bounded as well.

In the Maximally Balanced Connected Partition problem, the task is to partition a graph into two connected subgraphs such that the size of the smallest subgraph is maximised [23].

Another closely related problem is the Sparsest Cut problem asking whether there exists a cut with *sparsity* at most k, for some given $k \in \mathbb{Q}$. The sparsity of a cut $(S, \bar{S})$ is $\frac{|E(S,\bar{S})|}{\min\{|S|,|\bar{S}|\}}$, where $E(S, \bar{S})$ is the set of edges with endpoints in both S and $\bar{S}$. The problem is NP-hard [68] but there exists a polynomial-time approximation algorithm with ratio $O(\sqrt{\log n})$, with n the number of vertices in the graph.

Lastly, we mention the notion of *isoperimetric number* [20, 21] described as the minimum ratio $|N(X)|/|X|$ for all possible proper subsets of vertices X, with $N(X) = \left(\bigcup_{v \in X} N(v)\right) \setminus X$. Mohar proved that deciding if the isoperimetric number of a graph is smaller than a given constant is NP-complete [72].

Our definition of *proportionally dense subgraph* is closely related to the definition of a *community* as introduced in [73]. Olsen defines a *community structure* as a partition of the vertices into communities, where a part, *i.e.* an induced subgraph (with at least 2 vertices), is a *community* if and only if each vertex has proportionally as many neighbours in its community than in any other community. Hence, the notion of proportional density can be seen as a way to define dense regions of the graph, correlating the size of the region and the degree of each

vertex composing it. In [10], the authors investigate the notion of 2-community structure as a community structure with exactly two parts. We use the same definition (up to the special case where a community is of size one) to define a 2-PDS partition.

So far, only a few results are known about the existence of a 2-PDS partition in a graph, and the complexity of finding one. In fact, all known results are positive results, that is, polynomial-time algorithms to find a 2-PDS partition in a special class of graphs. It has been proved in [40] that deciding if a graph contains a 2-PDS partition with both PDS's of the same size is NP-complete. On trees [10, 40] and graphs with maximum degree 3 or minimum degree $n-3$ (n the order of the graph) a connected 2-PDS partition always exists and can be found in polynomial time [10]. The results extensively use the connectivity of the PDS's. To find a connected 2-PDS partition in a tree, one can prove that there exists an edge such that its removal yields two connected PDS's. If a graph has a maximum degree at most 3, a greedy algorithm keeps decreasing the size of a cut under some constraints and the removal of the final cut describes two connected PDS's.

Observe that, in the mentioned results, the algorithms do not *decide* whether there exists a 2-PDS partition but *find* and return a 2-PDS partition as solution. Since no graphs were known to not have a 2-PDS partition, we are left with a natural question: *Do graphs without a 2-PDS partition exist?* If the answer is *no*, that is, if all graphs were to admit a 2-PDS partition, one could consider the associated *search problem* and study its complexity. While the goal of this chapter is not to discuss the computational complexity of search problems (as we effectively answer *yes* to the latter question), it is worth mentioning that even if a decision problem is trivial (the existence is always true), its corresponding search problem may be hard to solve. Firstly introduced in [70], the class TFNP is the class of nondeterministic multivalued functions with values that are guaranteed to exist and can be verified in polynomial time. Simply put, TFNP is a complexity class that contains problems whose associated decision variants are trivial (the existence is always true). The class FP is a subclass of TFNP and contains problems (total functions) that can be solved in polynomial time. There is also the class FNP, superclass of TFNP that drops the requirement that the functions are total. Of course, $\mathsf{FP} \subseteq \mathsf{TFNP} \subseteq \mathsf{FNP}$. Deciding whether the inclusions are strict or not is equivalent to decide if $\mathrm{P} = \mathsf{NP}$. For further details on these complexity classes and their subclasses, we recommend the reading of [14, 19, 74, 75].

2.2. Proportional density and PDS's

Throughout the chapter, we use the notations introduced in Section 1.1. We also say that a vertex u is *universal* if $d(u) = |V| - 1$ and *pendant* if $d(u) = 1$.

Given a graph $G = (V, E)$, the density of a subgraph on a vertex set $S \subseteq V$ is usually defined as $\frac{|E(S)|}{|S|}$, where $E(S)$ is the set of edges in the subgraph. The problem of finding a subgraph of maximum density can be solved in polynomial time [51], but it becomes NP-hard when at least, or exactly, k vertices must belong to the subgraph [6, 42, 60].

Here, we introduce the notion of *proportional density*, which captures both the size of the subset and the number of neighbours. In turn, we define a *proportionally dense subgraph* (PDS) as an induced subgraph respecting the proportional density constraint.

Definition 2.1: Proportionally dense subgraph (PDS)

For a graph $G = (V, E)$, a *proportionally dense subgraph* of G is an induced subgraph on a vertex set $S \subset V$ such that each vertex $u \in S$ is *satisfied* in S, that is,

$$|\bar{S}| \cdot d_S(u) \geq (|S|-1) \cdot d_{\bar{S}}(u)\,, \text{ or, equivalently, } (|V|-1) \cdot d_S(u) \geq (|S|-1) \cdot d(u)\,.$$

Note that if $|S| \geq 2$, then we can rewrite the inequalities as

$$\frac{d_S(u)}{|S|-1} \geq \frac{d_{\bar{S}}(u)}{|\bar{S}|} \text{ or, equivalently, } \frac{d_S(u)}{|S|-1} \geq \frac{d(u)}{|V|-1}\,.$$

The proof of the equivalence can be found in [10]. Note that a subgraph containing a single vertex is also a PDS, but obviously a PDS cannot be the entire graph.

Definition 2.2: 2-PDS partition

A 2-*PDS partition* of a graph $G = (V, E)$ is a partition $\Pi = \{S_1, S_2\}$ of V such that S_1 and S_2 induce two PDS's in G.

In this chapter, we address the problem of deciding if a graph admits a 2-PDS partition. Notice that a PDS does not necessarily need to be connected. Therefore, we also consider the problem of deciding if a graph has a *connected* 2-*PDS partition*, that is, a 2-PDS partition whose PDS's are connected subgraphs. As we will see in Section 2.3, both problems are proved to be *decision problems* since we are

able to give an infinite class of graphs that do not admit a 2-PDS partition, and a second infinite class of graphs that only admit a disconnected 2-PDS partition, but not a connected one.

If a graph is disconnected, both problems become trivial, and hence we assume that all graphs are connected.

Overview of the results.
In Section 2.3.1, we construct an infinite family of graphs without a 2-PDS partition. As far as we know, these are the first negative results regarding the existence of a 2-PDS partition, and therefore the problem of deciding if a graph admits a 2-PDS partition is a *decision problem*. We also give examples of graphs without a 2-PDS partition that do not belong to the family. Then, in Section 2.3.2, we propose another infinite family of graphs without a connected 2-PDS but with a disconnected one.

2.3. Infinite classes of graphs

In the hope of finding small graphs without a 2-PDS partition, we used a computer based approach and generated all connected graphs up to 11 vertices[1] and checked for each graph whether it admits a 2-PDS partition, connected if possible. This procedure returned four graphs with 10 vertices (see Fig. 2.4) from which we were able to construct an infinite family of graphs without a 2-PDS partition. We also obtained graphs with 11 vertices that have a disconnected 2-PDS partition but no connected one. Again, from these examples we defined an infinite family of graphs which do not have a connected 2-PDS partition but admit a disconnected one. The structural properties of these examples were crucial to define our infinite families.

2.3.1. Graphs without a 2-PDS partition

The question about the existence of graphs without a 2-PDS was left open in [10]. To the best of our knowledge, no graphs without a 2-PDS partition were known. In this section we present an infinite class $\mathcal{G}$ (see Definition 2.3) of graphs with even number of vertices without a 2-PDS partition.

[1]For the generation of the graph, we used the program `geng` from the `gtools` suite [69]: `http://pallini.di.uniroma1.it/`

Definition 2.3

Let $\mathcal{G}$ be the class of graphs such that, if $G = (V, E) \in \mathcal{G}$, then

- $V = W_1 \cup W_2 \cup \{w, x, y, z\}$, where W_1, W_2 are cliques of the same size k, $k \geq 3$, and $\{w, x, y\}$ is a clique of size 3;
- w is adjacent to all vertices in $W_1 \cup W_2$, and z is only adjacent to y;
- $1 \leq d_{W_1}(x) = d_{W_2}(x) \leq k - 1$ and $2 \leq d_{W_1}(y) = d_{W_2}(y) \leq k - 1$;
- $|W_i \cap (N(x) \cup N(y))| > \frac{3k}{k+3}$ for each $i \in \{1, 2\}$;
- there exist vertices $\alpha, \beta \in W_1 \cup W_2$ such that $\alpha \in N(y) \setminus N(x)$, and $\beta \in N(x) \cap N(y)$;
- there is no edge between the vertex sets W_1 and W_2.

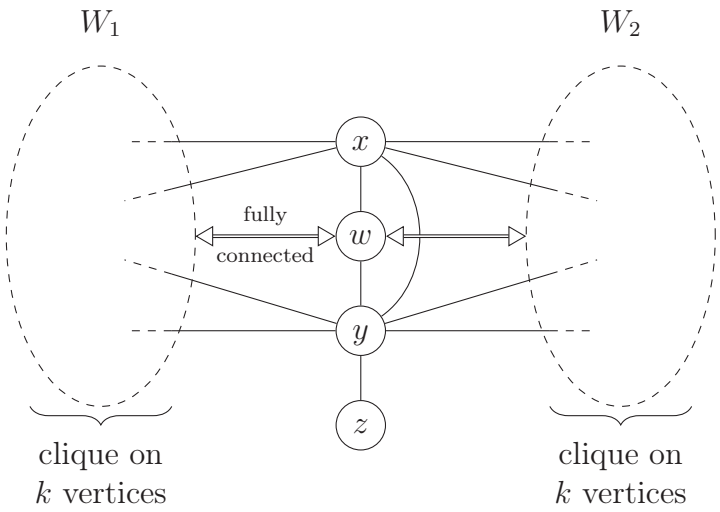


Figure 2.1: A schematic representation of a graph in $\mathcal{G}$.

Note that the smallest graphs in $\mathcal{G}$ have 10 vertices (see Fig. 2.2), and one of them is planar (see Fig. 2.3).

Theorem 2.1

All graphs in $\mathcal{G}$ do not have a 2-PDS partition.

Proof.

Let $G = (V, E)$ be a graph in $\mathcal{G}$. Firstly, notice that there is no 2-PDS partition $\{A, B\}$ in G such that $|A| = 1$ or $|B| = 1$. Without loss of generality, suppose by contradiction that $A = \{v\}$ for some vertex $v \in V$, and notice that the neighbour of v in B must be a universal vertex in order

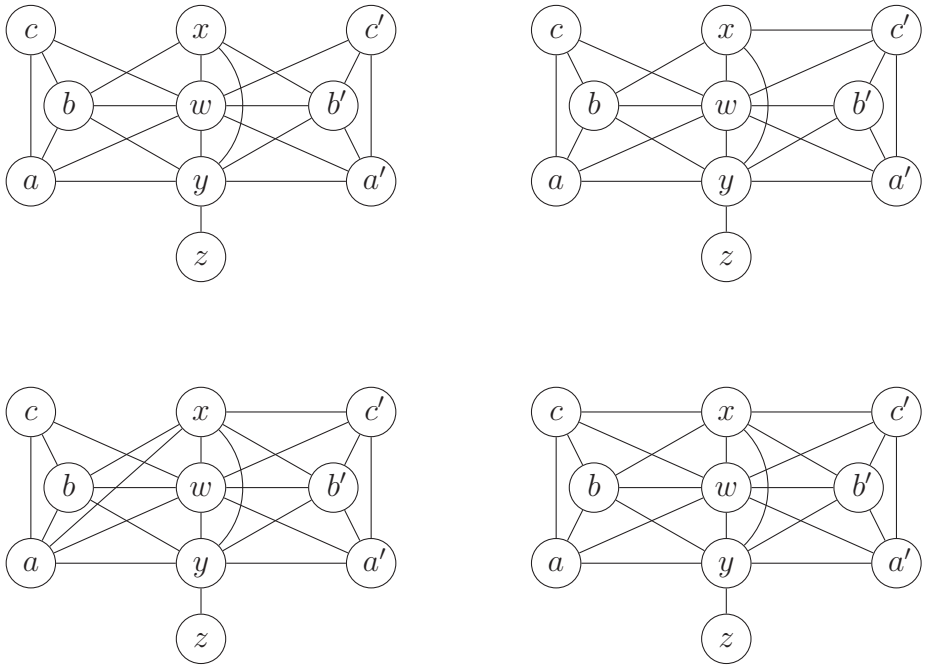


Figure 2.2: The four graphs with 10 vertices that belong to $\mathcal{G}$.

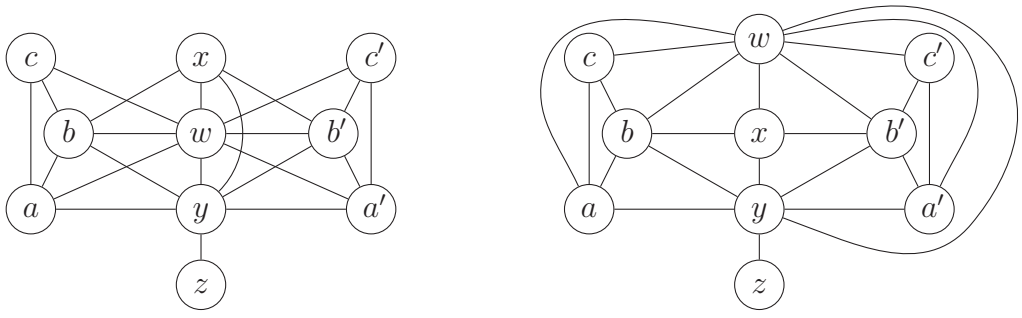


Figure 2.3: A planar graph from $\mathcal{G}$ with 10 vertices without a 2-PDS partition. On the left, its schematic representation as in Fig. 2.1; on the right, its planar representation.

to be satisfied. Since G does not contain a universal vertex, there is no 2-PDS partition $\{A, B\}$ in G with $|A| = 1$ or $|B| = 1$. Hence, assume that $|A|, |B| \geq 2$.

Observe that the vertex z is satisfied if and only if it belongs to the same PDS as the vertex y. Hence, without loss of generality, we assume that $y, z \in B$. In addition, the vertex w has degree $|V| - 2$ and is not connected to $z \in B$. Hence, necessarily $w \in A$.

Now we prove that for any partition $\{A, B\}$ of V, where $w \in A$ and $y, z \in B$, there is at least one vertex which is not satisfied, and hence there is no 2-PDS partition in G. For any partition $\{A, B\}$ of V, we denote by A_i and B_i the sets $A \cap W_i$ and $B \cap W_i$, respectively, for $i \in \{1, 2\}$. We split the proof into two cases: In the first case, we suppose that B_1 or B_2 is empty; in the second case, we assume that B_1 and B_2 are not empty.

Case 1: $B_1 = \emptyset$ or $B_2 = \emptyset$.

Suppose first that $B_1 = \emptyset$ and $B \subseteq \{x, y, z\} \cup W_2$.

- If $B_2 = \emptyset$, we have two possibilities:
  - if $x \in B$, then $B = \{x, y, z\}$ and $\beta \in A$ is not satisfied since $\frac{d_A(\beta)}{|A|-1} = \frac{k}{2k} < \frac{2}{3} = \frac{d_B(\beta)}{|B|}$;
  - if $x \in A$, then $B = \{y, z\}$ and $\alpha \in A$ is not satisfied since $\frac{d_A(\alpha)}{|A|-1} = \frac{k}{2k+1} < \frac{1}{2} = \frac{d_B(\alpha)}{|B|}$.
- If $B_2 \neq \emptyset$ and $B_2 \neq W_2$:
  - Case $x \in B$.
    - If there exists $u \in A_2$ such that $u \in N(x) \cup N(y)$ and u is satisfied, then we have:

$$\frac{|A_2|}{k + |A_2|} = \frac{d_A(u)}{|A| - 1} \geq \frac{d(u)}{|V| - 1} \geq \frac{k+1}{2k+3},$$

      which implies that $|A_2| \cdot (k+2) \geq k \cdot (k+1)$, and hence that $|A_2| > k - 1$. A contradiction since $|A_2| \leq k - 1$.
    - Otherwise, for all $u \in A_2$, $u \notin N(x) \cup N(y)$. Hence, for any $u \in A_2$, if u is satisfied then:

$$\frac{|A_2|}{k + |A_2|} = \frac{d_A(u)}{|A| - 1} \geq \frac{d(u)}{|V| - 1} = \frac{k}{2k+3},$$

which implies that $|A_2| \cdot (k+3) \geq k^2$, and hence that $|A_2| \geq \frac{k^2}{k+3}$. Due to our assumptions about the graph, $|W_2 \cap (N(x) \cup N(y))| > \frac{3k}{k+3}$. Thus, $k - \frac{3k}{k+3} > |W_2 \setminus (N(x) \cup N(y))| \geq |A_2| \geq \frac{k^2}{k+3}$ which implies $k > k$, a contradiction.

- Case $x \in A$. Let $u \in A_2$.
  - If $u \in N(y) \cap N(x)$ and u is satisfied, then we have:

$$\frac{|A_2|+1}{k+|A_2|+1} = \frac{d_A(u)}{|A|-1} \geq \frac{d(u)}{|V|-1} = \frac{k+2}{2k+3},$$

    which implies that $|A_2| \geq k - \frac{1}{k+1}$, and then $|A_2| \geq k$, a contradiction since $B_2 \neq \emptyset$.
  - If $u \in N(y) \setminus N(x)$, then $d_A(u) = |A_2|$ and $d(u) = k+1$. Therefore, similarly to the previous case, we obtain that $|A_2| \geq k + \frac{1}{k+2}$ and so $|A_2| > k$, a contradiction.
  - If $u \in N(x) \setminus N(y)$, then:

$$\frac{|A_2|+1}{k+|A_2|+1} = \frac{d_A(u)}{|A|-1} \geq \frac{d(u)}{|V|-1} = \frac{k+1}{2k+3},$$

    which implies that $|A_2| \cdot (k+2) \geq k^2 - 2$, and hence $|A_2| \geq \frac{k^2-2}{k+2} > k-2$. Since assuming that there is a vertex in $A_2 \cap N(y)$ leads to a contradiction (see previous cases), we can assume that $A_2 \cap N(y) = \emptyset$. Then, since $d_{W_2}(y) \geq 2$, then $|W_2 \setminus N(y)| \leq k-2$. Thus, $k-2 \geq |A_2| > k-2$, a contradiction.
  - If $u \notin N(x) \cup N(y)$, then $d_A(u) = |A_2|$ and $d(u) = k$. Again, similarly to the previous case, we obtain $|A_2| > k-2$ and a contradiction.

- If $B_2 = W_2$, then either $B = \{x, y, z\} \cup W_2$, and we have $|A|+2 = |B|$ but $d_A(x) = d_B(x)$, and thus x is not satisfied, or $B = \{y, z\} \cup W_2$; since $|A| = |B|$ we have: $\frac{d_B(y)}{|B|-1} < \frac{d_B(y)+1}{|B|} = \frac{d_A(y)}{|B|} = \frac{d_A(y)}{|A|}$, and therefore y is not satisfied.

We conclude that if there is a 2-PDS partition in G, then $B_1 \neq \emptyset$. The case $B_2 = \emptyset$ is similar, and therefore if there is a 2-PDS partition in G, then $B_2 \neq \emptyset$.

Case 2: $B_1, B_2 \neq \emptyset$.

Without loss of generality, we suppose $|B_1| \leq |B_2|$. Let $u \in B_1$ and suppose that u is satisfied in the partition $\{A, B\}$. We prove that in all cases, if u is satisfied then it implies a contradiction with $|B_1| \leq |B_2|$.

* If $x \in A$:
  - If $u \in N(x) \cap N(y)$ is satisfied, then:

$$\frac{|B_1|}{|B_1| + |B_2| + 1} = \frac{d_B(u)}{|B| - 1} \geq \frac{d(u)}{|V| - 1} = \frac{k+2}{2k+3},$$

    which implies that $|B_1| \cdot (k+1) \geq (|B_2| + 1) \cdot (k+2)$, and hence that $|B_1| > |B_2|$. A contradiction with the assumption that $|B_1| \leq |B_2|$, and hence u is not satisfied.
  - If $u \in N(x) \setminus N(y)$, we have $d_B(u) = |B_1| - 1$ and $d(u) = k + 1$ and similarly we obtain $|B_1| \cdot (k+2) \geq |B_2| \cdot (k+1) + (3k+4) \geq |B_2| \cdot (k+1) + (|B_2| + 4) > |B_2| \cdot (k+2)$, a contradiction since $|B_1| \leq |B_2|$.
  - If $u \in N(y) \setminus N(x)$, we have $d_B(u) = |B_1|$ and $d(u) = k + 1$ and similarly we obtain $|B_1| \cdot (k+2) \geq |B_2| \cdot (k+1) + (k+1) \geq |B_2| \cdot (k+1) + (|B_2| + 1) > |B_2| \cdot (k+2)$, a contradiction since $|B_1| \leq |B_2|$.
  - If $u \notin N(x) \cup N(y)$, we have $d_B(u) = |B_1| - 1$ and $d(u) = k$ and similarly we obtain $|B_1| \cdot (k+3) \geq |B_2| \cdot k + 3(k+1) \geq |B_2| \cdot k + 3(|B_2| + 1) > |B_2| \cdot (k+3)$, a contradiction since $|B_1| \leq |B_2|$.

* If $x \in B$:
  - If $u \in N(x) \cap N(y)$ is satisfied, then:

$$\frac{|B_1| + 1}{|B_1| + |B_2| + 2} = \frac{d_B(u)}{|B| - 1} \geq \frac{d(u)}{|V| - 1} = \frac{k+2}{2k+3},$$

    which implies that $|B_1| \cdot (k+1) \geq |B_2| \cdot (k+2) + 1$, and thus that $|B_1| > |B_2|$. A contradiction with the assumption that $|B_1| \leq |B_2|$, and hence u is not satisfied.
  - If $u \in N(x) \setminus N(y)$ or $u \in N(y) \setminus N(x)$, we have $d_B(u) = |B_1|$ and $d(u) = k + 1$ and similarly we obtain $|B_1| \cdot (k+2) \geq |B_2| \cdot (k+1) + 2(k+1) \geq |B_2| \cdot (k+1) + 2(|B_2| + 1) > |B_2| \cdot (k+3)$, a contradiction since $|B_1| \leq |B_2|$.

– If $u \notin N(x) \cup N(y)$, we have $d_B(u) = |B_1| - 1$ and $d(u) = k$ and similarly we obtain $|B_1| \cdot (k+3) \geq |B_2| \cdot k + 4k + 3 \geq |B_2| \cdot k + 4 \cdot |B_2| + 3 > |B_2| \cdot (k+4)$, a contradiction since $|B_1| \leq |B_2|$.

■

In Fig. 2.4, we present four graphs with 11 vertices without a 2-PDS partition. These graphs have an odd number of vertices, and hence they do not belong to $\mathcal{G}$. To prove that they do not have a 2-PDS partition, one can notice that, like the graphs in $\mathcal{G}$, they have a pendant vertex z connected to a vertex y, and a vertex w connected to all the vertices except the pendant vertex. As a result, the vertex z is satisfied if and only if it belongs to the same PDS as y, and thus w must be in the other PDS. The rest of the proof can be done by case distinction.

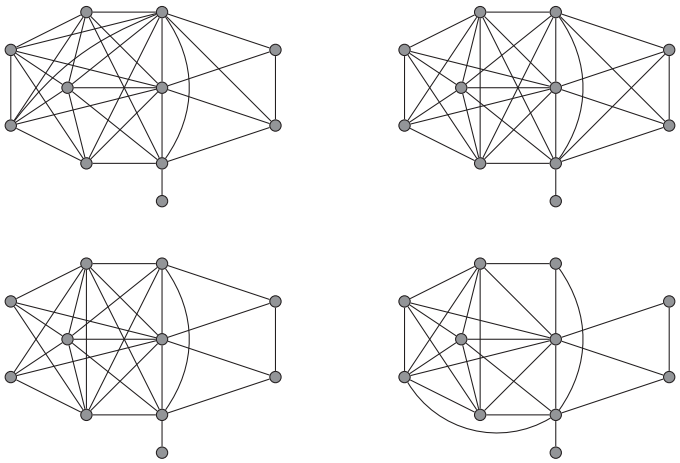


Figure 2.4: Four graphs with 11 vertices which do not have a 2-PDS partition

2.3.2. Graphs without a connected 2-PDS partition

Now, we present an infinite family of graphs where each graph admits a disconnected 2-PDS partition, but not a connected one. The existence of such graphs was left as an open problem in [10].

Definition 2.4

We define the infinite class of graphs $\mathcal{H}$ such that, if $G = (V, E) \in \mathcal{H}$, then

- $V := W \cup \{\alpha_1, \beta_1, \alpha_2, \beta_2\}$, where W is a clique of odd size at least 7;
- $\exists x \in W$ such that $N(x) \setminus W := \{\alpha_1, \beta_1, \beta_2\}$;

- $\exists y \in W$ such that $N(y) \setminus W := \{\alpha_2, \beta_2, \beta_1\}$;
- $N(\alpha_1) := \{\beta_1, x\}$, $N(\alpha_2) := \{\beta_2, y\}$;
- $N(\beta_1) := \{\alpha_1, x, y\}$, $N(\beta_2) := \{\alpha_2, x, y\}$.

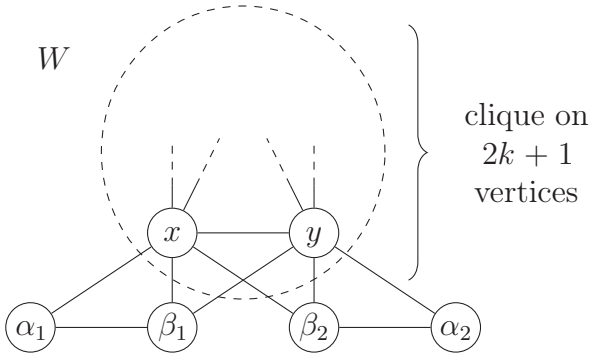


Figure 2.5: A schematic representation of a graph in $\mathcal{H}$.

Compared to the graphs in $\mathcal{G}$, each graph in $\mathcal{H}$ has an odd number of vertices, the smallest one has 11 vertices.

Theorem 2.2

All graphs in $\mathcal{H}$ do not have a connected 2-PDS partition but have a disconnected one.

Proof. Let $G = (V, E) \in \mathcal{H}$. Suppose that G has a connected 2-PDS partition $\{A, B\}$. If $A \subseteq W$, then we have two cases: either $A = W$ but then $G[B]$ is disconnected or $A \subset W$ but then a vertex in $W \setminus A$ is not satisfied in B. Hence, $A \nsubseteq W$ and similarly $B \nsubseteq W$. Consequently, to guarantee the connectivity of $G[A]$ and $G[B]$, the vertices x and y must be in different parts of the partition. Therefore, we assume without loss of generality that $x \in A$ and $y \in B$.

If $\alpha_1 \in B$, then y is not satisfied since it is connected to each vertex in A. Similarly, α_2 cannot belong to A since otherwise x is not satisfied. As a result, we only have to consider the possible cases for β_1 and β_2, knowing that $x, \alpha_1 \in A$ and $y, \alpha_2 \in B$.

If $A = \{x, \alpha_1, \beta_1\}$, then the vertex β_2 is not satisfied in B since $\frac{d_B(\beta_2)}{|B|-1} \leq \frac{2}{7} < \frac{1}{3} = \frac{d_A(\beta_2)}{|A|}$. Similarly, if $B = \{y, \alpha_2, \beta_2\}$, then the vertex β_1 is not satisfied in A.

If $\{x, \alpha_1, \beta_1\} \subset A$ and $\{y, \alpha_2, \beta_2\} \subset B$, then consider two vertices $a \in (W \setminus \{x\}) \cap A$ and $b \in (W \setminus \{y\}) \cap B$. The vertex a is satisfied in A if and only if

$$\frac{d_A(a)}{|A|-1} = \frac{|A|-3}{|A|-1} \geq \frac{|B|-2}{|B|} = \frac{d_B(a)}{|B|},$$

which implies that $|A| \geq |B| + 1$. Similarly, the vertex b is satisfied in B if and only if $|A| \leq |B| - 1$, which is a contradiction.
If $\beta_1, \beta_2 \in A$, then the vertex β_2 is satisfied in A if and only if

$$\frac{d_A(\beta_2)}{|A|-1} = \frac{1}{|A|-1} \geq \frac{2}{|B|} = \frac{d_B(\beta_2)}{|B|},$$

which implies that $|A| \leq \frac{|B|}{2} + 1$. Moreover, the vertex α_2 is satisfied in B if and only if

$$\frac{d_B(\alpha_2)}{|B|-1} = \frac{1}{|B|-1} \geq \frac{1}{|A|} = \frac{d_A(\alpha_2)}{|A|},$$

which implies that $|A| \geq |B| - 1$. We then obtain $|B| - 1 \leq |A| \leq \frac{|B|}{2} + 1$, and therefore $|B| \leq 4$. Thus, $|A| \leq 3$, which is not possible since $|V| \geq 11$. Similar arguments can be used to prove that β_1 and β_2 cannot both belong to B.
We conclude that G does not have a connected 2-PDS partition. However, it is easy to see that, if $A := \{\alpha_1, \beta_1, \alpha_2, \beta_2\}$ and $B := V \setminus A$, then $\{A, B\}$ is a disconnected 2-PDS partition of G. ■

2.4. Conclusion and further work

The notion of proportional density, and the definition of a proportionally dense subgraph, is based on a combination of local and global properties: In a PDS each vertex has to satisfy a condition depending not only on its degree but also on the size of the subgraph. This condition makes the problem complex from an algorithmic point of view and requires a novel approach.

Our infinite families of graphs bring a new insight into the existence of 2-PDS partitions in graphs, with and without constraint of connectivity. One way of research could try to find a characterisation of these graphs. Obviously, the main open question is the complexity of deciding the existence of a (connected) 2-PDS partition.

Chapter 3

Max Proportionally Dense Subgraph

Outline

Motivated by the notion of proportional density and PDS studied in the previous chapter, we now investigate the complexity of the problem of finding a PDS of maximum size in a graph. We prove several hardness results for the problem on split and bipartite graphs. On the positive side, we present a polynomial-time approximation algorithm for the problem. We also show that all Hamiltonian cubic graphs with n vertices (except two) have a PDS of size $\lfloor \frac{2n+1}{3} \rfloor$, which we prove to be an upper bound on the size of a PDS in cubic graphs.

Some of the results presented in this chapter appear in the following paper:

- C. Bazgan, J. Chlebíková, C. Dallard and T. Pontoizeau, 'Proportionally dense subgraph of maximum size: complexity and approximation', *arXiv e-prints*, Accepted for publication in *Discrete Applied Mathematics*, 2019. DOI: 10.1016/j.dam.2019.07.010. arXiv: 1903.06579 [cs.CC].

3.1. Introduction

In Chapter 2, we have defined the notions of *proportional density* and *proportionally dense subgraph* (PDS). In this chapter, we use the same notions but focus on the problem of finding a PDS of maximum size in a graph.

For a graph $G = (V, E)$, the density of a subgraph on a vertex set $S \subseteq V$ is commonly defined as $\frac{|E(S)|}{|S|}$, where $E(S)$ is the set of edges in the subgraph. The problem of finding a subgraph of maximum density can be solved in polynomial time using a max flow technique [51]. However, when the subgraph must contain *exactly* k vertices, the problem becomes NP-hard [6, 42] and is known as the Densest k-subgraph problem. Two variants of the problem have also been studied where the number of vertices in the subgraph must be either *at least* k or *at most* k. The former is known to be NP-hard [60], but there exists a polynomial-time 2-approximation algorithm to solve it [5]. It was also shown that any α-approximation for the *at most* k variant would imply a $\Theta(\alpha^2)$-approximation for the densest k-subgraph problem [5].

As we have seen in Chapter 2, an induced subgraph on a vertex set $S \subset V$ is said to be *proportionally dense* if all of its vertices in S have *proportionally* as many neighbours in the subgraph as in the graph, hence the condition $\frac{d_S(u)}{|S|-1} \geq \frac{d(u)}{|V|-1}$ holds for each vertex u in S. A *proportionally dense subgraph* grants more importance to the vertices than the standard definition of a dense subgraph, as all the vertices in a PDS must be 'satisfied', *i.e.* respect the above condition. This can be compared with *defensive alliances* in graphs, where the vertices in the alliance must have at least as many neighbours inside the alliance than outside it [61, 79], without the notion of proportion of neighbours.

From a theoretical point of view, it is interesting to observe a problem that connects local and global properties of vertex subsets, interweaving the size of the subset and the number of neighbours. Moreover, this paradigm has rarely been seen in graph theory problems.

Overview of the results.
In Section 3.2, we introduce the basic notations used in the chapter. Section 3.3 presents various hardness results of the Max Proportionally Dense Subgraph problem. Then, we give positive results about the approximation of this problem in Section 3.4. We also prove that the the problem can be solved in linear time on Hamiltonian cubic graphs in Section 3.5. Conclusion and open problems are given in Section 3.6.

3.2. Notations and definitions

Throughout the chapter, we use the notations introduced in Section 1.1. We say that a graph G is a *Hamiltonian graph* if it contains a *Hamiltonian cycle* (a cycle going through all the vertices). Moreover, G is a *cubic graph* if $d(u) = 3$ for any vertex $u \in V$. Lastly, G is a *split graph* if the vertex-set can be partitioned into a clique and an independent set.

We assume that all graphs contain at least 3 vertices.

Although Definition 2.1 from Chapter 2 already defines the notion of PDS, for the sake of completeness, let us recall it.

Definition 3.1: Proportionally dense subgraph (PDS)

For a graph $G = (V, E)$, a *proportionally dense subgraph* of G is an induced subgraph on a vertex set $S \subset V$ such that each vertex $u \in S$ is *satisfied* in S, that is,

$$|\bar{S}|\cdot d_S(u) \geq (|S|-1)\cdot d_{\bar{S}}(u)\,, \text{ or, equivalently, } (|V|-1)\cdot d_S(u) \geq (|S|-1)\cdot d(u)\,.$$

Note that if $|S| \geq 2$, then we can rewrite the inequalities as

$$\frac{d_S(u)}{|S|-1} \geq \frac{d_{\bar{S}}(u)}{|\bar{S}|} \text{ or, equivalently, } \frac{d_S(u)}{|S|-1} \geq \frac{d(u)}{|V|-1}\,. \tag{3.1}$$

Note that two adjacent vertices form a PDS of size 2. Therefore, in the rest of the chapter we assume that every PDS contains at least 2 vertices and make use of the ratios in the inequality.

Max PDS

Input: A graph G.
Output: A proportionally dense subgraph in G of maximum size.

Remark 3.1. Observe that a proportionally dense subgraph may be connected or not, even if the graph is connected. We study both cases and talk about a *connected PDS* in the former case.

To give an example of graphs illustrating Remark 3.1, we give two connected graphs in which the proportionally dense subgraph of maximum size is not connected. In the cubic graph in Fig. 3.1, the grey vertices represent a PDS of size 7, which is not connected. In fact, for any set $S \subset V$, at least one vertex $u \in S$ has $d_S(u) \leq 2$. If S is a PDS, then we must have $\frac{2}{|S|-1} \geq \frac{d_S(u)}{|S|-1} \geq \frac{d(u)}{|V|-1} = \frac{3}{9} = \frac{1}{3}$,

which directly implies that $|S| \leq 7$. Morever, if the set S contains 7 vertices and induces a connected subgraph, then there exists $u \in S$ such that $d_S(u) = 1$, which is not satisfied since $\frac{1}{7-1} = \frac{d_S(u)}{|S|-1} < \frac{d_{\bar{S}}(u)}{|\bar{S}|} = \frac{2}{3}$. It can be checked that the maximum size for a connected PDS is only 5. Similarly, in the caterpillar in Fig. 3.1, any connected induced subgraph of size at least 12 has one unsatisfied vertex. The maximum size for a PDS is 12, while only 8 for a connected PDS.

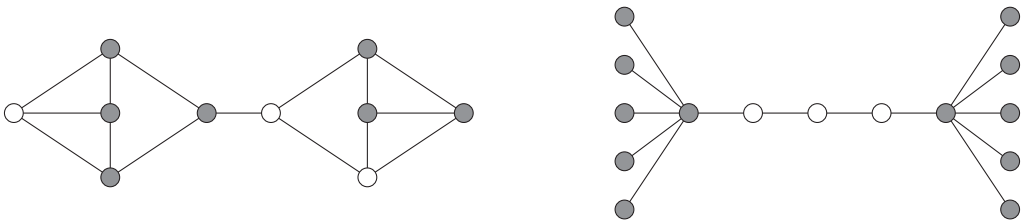


Figure 3.1: Two connected graphs in which all PDS of maximum size are not connected. Gray vertices represent a PDS of maximum size in each graph.

3.3. Hardness results

In this section we prove several hardness results for Max PDS on split and bipartite graphs, and further extend the results to prove that deciding if a PDS is inclusion wise maximal is coNP-complete.

We construct two polynomial-time reductions from Max Independent Set, which is known to be NP-hard [57].

Max Independent Set

Input: A graph G.
Output: An independent set in G of maximum size.

3.3.1. Split graphs

We first describe a polynomial-time reduction, and then prove two intermediate results allowing us to prove the NP-hardness of Max PDS on split graphs.

Construction 3.1

Let $G = (V, E)$ be a connected graph not isomorphic to a star. We define the graph $G' = (V', E')$ as follows:

- $V' := \{z_1, z_2\} \cup M \cup N$, where $N := V$, $M := \{uv : \{u, v\} \in E\}$ and z_1, z_2 are two additional vertices;
- for all $e \in M$ and $u \in N$, the edge $\{e, u\} \in E'$ if and only if $u \notin e$;
- the set $M \cup \{z_1, z_2\}$ induces a clique in G'.

Obviously, Construction 3.1 can be done in polynomial time. Notice that G' is a split graph, and is connected if and only if G is not isomorphic to a star. See Fig. 3.2 for an example.

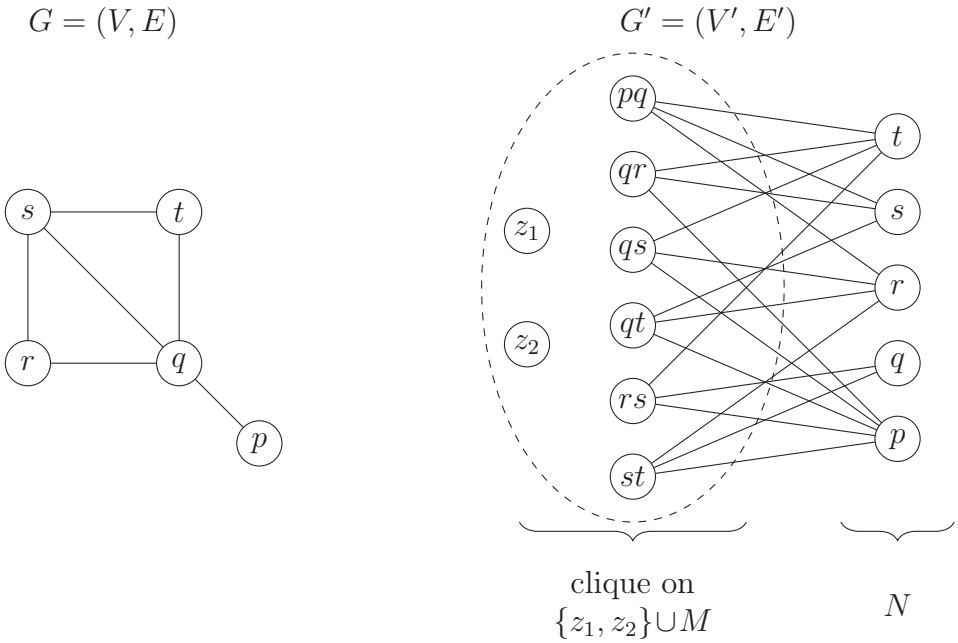


Figure 3.2: The graph G' obtained from the graph G using Construction 3.1.

Lemma 3.1

Let $G = (V, E)$ be a graph not isomorphic to a star and let $G' = (V', E')$ be the graph obtained through Construction 3.1. Let $S \subset V'$ be a set of vertices such that $M \cup \{z_1, z_2\} \subseteq S$. Then a vertex $e \in M$ is satisfied in $G'[S]$ if and only if $d_S(e) \geq |S| - 2$.

Proof. A vertex $e \in M$ has degree $d(e) = |V'| - 3$. Hence if $d_S(e) < |S| - 2$, then $d_{\bar{S}}(e) = |\bar{S}|$, and thus e is not satisfied in $G'[S]$. However, if $d_S(e) \geq |S| - 2$, then $d_{\bar{S}}(e) < |\bar{S}|$. Also, since G is connected, then $|M| \geq |N| - 1$ and $|S| \geq |M| + 2 > |N| \geq |\bar{S}|$. Therefore, we have

$$|\bar{S}| \cdot d_S(e) \geq |\bar{S}| \cdot (|S| - 2) \geq (|\bar{S}| - 1) \cdot (|S| - 1) \geq (|S| - 1) \cdot d_{\bar{S}}(e)\,,$$

and thus e is satisfied in $G'[S]$. ■

Lemma 3.2

Let $G = (V, E)$ be a graph not isomorphic to a star and let $G' = (V', E')$ be the graph obtained through Construction 3.1. Let $S_1 \subset V'$ such that $G'[S_1]$ is a PDS. Then there exists $S_2 \subset V'$ such that $G'[S_2]$ is a PDS, $|S_2| \geq |S_1|$ and $M \cup \{z_1, z_2\} \subseteq S_2$. Moreover, S_2 can be found in polynomial time.

Proof. Firstly, we show that $N \nsubseteq S_1$.

- if $S_1 = N$, since $G'[N]$ is an independent set, then any vertex $u \in S_1$ has $d_{S_1}(u) = 0$ and $d_{\bar{S}_1}(u) > 0$; hence u is not satisfied and $G'[S_1]$ is not a PDS;
- if $N \subset S_1$, then $\bar{S}_1$ is a subset of the clique $M \cup \{z_1, z_2\}$; it means any vertex $u \in S_1 \cap (M \cup \{z_1, z_2\})$ has $d_{\bar{S}_1}(u) = |\bar{S}_1|$ and $d_{S_1}(u) < |S_1| - 2$, and thus

  $$|\bar{S}_1| \cdot d_{S_1}(u) < |\bar{S}_1| \cdot (|S_1| - 2) < |\bar{S}_1| \cdot (|S_1| - 1) = (|S_1| - 1) \cdot d_{\bar{S}_1}(u)\,,$$

  so u is not satisfied and $G'[S_1]$ is not a PDS.

Now, let $S_2 := S_1 \cup M \cup \{z_1, z_2\}$ and $\bar{S}_2 := V' \setminus S_2$.
Observe that for any $f \in S_1 \cap M$, $d_{S_2}(f) - d_{S_1}(f) = |S_2| - |S_1| \geq 0$ and $d_{\bar{S}_2}(f) \leq d_{\bar{S}_1}(f)$. Thereby, we obtain $\frac{d_{S_2}(f)}{|S_2|-1} \geq \frac{d_{S_1}(f)}{|S_1|-1} \geq \frac{d_{\bar{S}_1}(f)}{|\bar{S}_1|} \geq \frac{d_{\bar{S}_2}(f)}{|\bar{S}_2|}$, so f is satisfied in S_2. Also, if a vertex in M is satisfied in S_2, then according to Lemma 3.1 it is also satisfied in any $S_2' \subseteq S_2$, as long as $M \cup \{z_1, z_2\} \subseteq S_2'$.
If there exists $e \in M \setminus S_1$ which is not satisfied in S_2, then following Lemma 3.1 it holds $d_{S_2}(e) < |S_2| - 2$. Thus, there exists a vertex $u \in S_2 \cap N$, non-adjacent to e, which we can transfer from S_2 to $\bar{S}_2$. Obviously, at most $|M \setminus S_1|$ transfers are needed to satisfy all the vertices in S_2, and thus $|S_2| \geq |S_1|$ holds true. Since $S_2 \cap N \subseteq S_1 \cap N$ and $N \nsubseteq S_1$, then $S_2 \neq V'$.
Note that $\bar{S}_2 \subseteq N$ and that each vertex $u \in S_1 \cap N$ is satisfied in S_2, since $d_{\bar{S}_2}(u) = 0$. Clearly, z_1 and z_2 are satisfied in S_2. Thus, $G'[S_2]$ is a PDS, and it can be found in polynomial time. ■

Notice that Lemma 3.2 implies that there exists a PDS of maximum size in G' that is connected. Hence, the following result also holds when looking for a connected PDS.

Theorem 3.1

MAX PROPORTIONALLY DENSE SUBGRAPH is NP-hard on split graphs.

Proof. Let $G = (V, E)$ be a graph not isomorphic to a star, $G' = (V', E')$ be the graph obtained through Construction 3.1, and $k \in \{1, \ldots, |V| - 2\}$. Notice that since G is connected and not isomorphic to a star, then there is no independent set of size $|V|-1$ in G. We claim that there is an independent set of size at least k in G if and only if there is a PDS of size at least $|M| + 2 + k$ in G'.

Let R be an independent set of G of size at least k. In G', we define $S := M \cup \{z_1, z_2\} \cup R$ and $\bar{S} := V' \setminus S$. First, note that $R \subseteq N$ thus $\bar{S} = N \setminus R$. The vertices in $S \cap N \cup \{z_2, z_2\}$ are obviously satisfied in $G'[S]$ as they only have neighbours in S. Hence, if there exist unsatisfied vertices, then they must be from the set M. Choose a vertex $e \in M$. Since R is an independent set of G, then for each edge $e = \{u, v\} \in E$ at most one of the vertices u and v belongs to R. Hence, the vertex $e \in M$ is not adjacent to at most one vertex in S and thus $d_S(e) \geq |S| - 2$. According to Lemma 3.1, the vertex e is satisfied in $G[S]$. Thus $G[S]$ is a PDS of size at least $|M| + 2 + k$. Let $S \subset V'$ be of size at least $|M|+2+k$ such that $G'[S]$ is a PDS. According to Lemma 3.2, there exists $S' \subset V'$ such that $G'[S']$ is a PDS, $|S'| \geq |S|$ and $\{z_1, z_2\} \cup M \subseteq S'$. We claim that $R' := S' \cap N$ is an independent set of G of size at least k. Obviously $|R'| \geq k$. Moreover, Lemma 3.1 states that for all satisfied vertices $e \in M$, $d_{S'}(e) \geq |S'| - 2$. This means that for each vertex $e \in M$ there is at most one vertex $u \in S'$ that is not adjacent to e. Since the vertices $e \in M$ and $u \in N$ are not adjacent in G', it implies that $u \in e$ in G, and therefore the edge $e \in E$ has at most one endpoint $u \in R'$ in the graph G. Thus, R' is an independent set of size at least k. ■

There exist several ways to prove that a problem is APX-hard. Below, we propose two different proofs showing that MAX PDS is APX-hard on split graphs.

In Proposition 3.1, we show that the reduction used in Theorem 3.1 is an approximation-preserving reduction, more specifically an L-reduction from MAX INDEPENDENT SET to MAX PDS. In turn, we obtain that MAX PDS is APX-hard, even on split graphs. Then, in Proposition 3.2, we use the concept of gap-preserving reduction to show that it is NP-hard to approximate MAX PDS within 1.0026028 on split graphs. This clearly implies the APX-hardness of MAX PDS on split graphs.

Proposition 3.1

MAX PROPORTIONALLY DENSE SUBGRAPH is APX-hard on split graphs.

Proof. We prove that the reduction from Theorem 3.1 can be seen as an L-reduction (see Definition 1.14) from MAX INDEPENDENT SET on cubic graphs to MAX PDS. Let I be an instance of MAX INDEPENDENT SET on the cubic graph $G = (V, E)$. We construct an instance I' of MAX PDS defined on the graph $G' = (V', E')$ obtained through Construction 3.1.
Since each cubic graph is 4-colourable, $OPT(I) \geq \frac{|V|}{4}$. Moreover, a cubic graph has exactly $\frac{3|V|}{2}$ edges, and hence $OPT(I') = 2 + |E| + OPT(I) = 2 + \frac{3|V|}{2} + OPT(I) \leq 2 + \frac{3|V|}{2} + |V| \leq 2 + \frac{5|V|}{2} \leq 2 + 10 \cdot OPT(I) \leq 12 \cdot OPT(I)$.
For any S inducing a PDS in G' we can construct an independent set R in G of size $|R| = |S| - |E| - 2$. Since $OPT(I') \geq |S| = |R| + |E| + 2$, in particular, when R is a maximum independent set, $OPT(I') \geq OPT(I) + |E| + 2$. In addition, $OPT(I) \geq |R| = |S| - |E| - 2$, and when $G'[S]$ is a maximum PDS, $OPT(I) \geq OPT(I') - |E| - 2$. Thus, $OPT(I) = OPT(I') - |E| - 2$ and $OPT(I) - |R| = OPT(I') - |S|$.
Since MAX INDEPENDENT SET is APX-hard on cubic graphs [2], we conclude that MAX PDS is APX-hard on split graphs. ■

Proposition 3.2

It is NP-hard to approximate MAX PROPORTIONALLY DENSE SUBGRAPH within 1.0026028 on split graphs, and hence the problem is APX-hard (even on split graphs).

Proof. Let I be an instance of MAX INDEPENDENT SET on a cubic graph $G = (V, E)$. It is known that it is NP-hard to decide whether $OPT(I) < \frac{12\tau+11+2\varepsilon}{24\tau+28} \cdot |V|$ or $OPT(I) > \frac{12\tau+12-2\varepsilon}{24\tau+28} \cdot |V|$, for any $\varepsilon > 0$, where $\tau \leq 6.9$ [22].
We construct an instance I' of MAX PDS defined on the graph $G' = (V', E')$ obtained through Construction 3.1. Note that $M \subset V'$ is of size $|E|$, that is $|M| = |E| = \frac{3|V|}{2}$ since G is cubic. From Theorem 3.1, we know that $OPT(I') = |M| + 2 + OPT(I)$. Consequently, it is NP-hard to decide whether $OPT(I') < |M| + 2 + \frac{12\tau+11+2\varepsilon}{24\tau+28} \cdot |V| = \frac{48\tau+53+2\varepsilon}{24\tau+28} \cdot |V| + 2$ or $OPT(I') > |M| + 2 + \frac{12\tau+12-2\varepsilon}{24\tau+28} = \frac{48\tau+54-2\varepsilon}{24\tau+28} \cdot |V| + 2$. We obtain that it is NP-hard to approximate MAX PDS within 1.0026028. ■

3.3.2. Bipartite graphs

In the following, we modify the previous construction in order to prove the NP-hardness of Max PDS on bipartite graph. The reduction will also be used to show the NP-hardness of an "extension version" of the problem, implying the coNP-completeness of deciding if a PDS is inclusion wise maximal.

Construction 3.2

Let $G = (V, E)$ be a graph with $|E| \geq |V|$, and an integer k such that $1 \leq k < |V| - 1$. We define the graph $G'_k = (V', E')$ as follows:

- $V' := L \cup M \cup N$, where $N := V$, $M := \{uv : \{u, v\} \in E\}$ and L contains $|L| := |M| \cdot (|V| - k - 1) - k + 1$ additional vertices;
- for all $e \in M$ and $u \in N$, the edge $\{e, u\} \in E'$ if and only if $u \notin e$;
- for all $e \in M$ and $v \in L$, the edge $\{e, v\} \in E'$.

Obviously, Construction 3.2 can be done in polynomial time. Clearly, G'_k is connected if and only if the input graph is not isomorphic to a star, which can't be since $|E| \geq |V|$. Also, notice that G'_k is a bipartite graph as there are edges only between M and $L \cup N$. See Fig. 3.3 for an example.

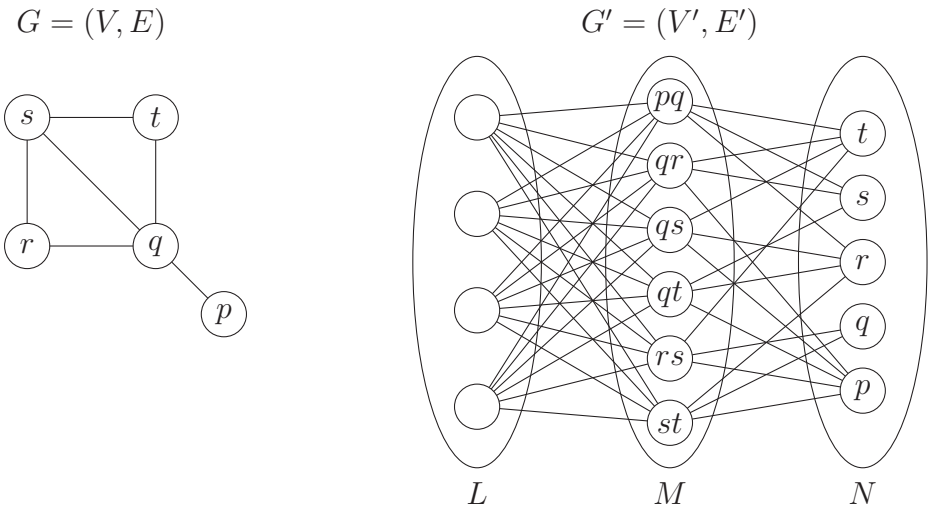


Figure 3.3: The graph G'_k obtained from G through Construction 3.2 using with $k = 3$.

We now prove intermediate results, which help concluding that Max PDS is NP-complete on bipartite graphs.

Lemma 3.3

Let m, n and k be integers such that $1 \leq k < n-1 < m$ and $\ell := m \cdot (n-k-1) - k + 1$. Then $\frac{\ell+k-1}{\ell+m+k-1} = \frac{n-k-1}{n-k}$.

Proof. $(n-k)\cdot(\ell+k-1) = (n-k-1)\cdot(\ell+k-1)+\ell+k-1 = (n-k-1)\cdot(\ell+k+1)+m\cdot(n-k-1) = (n-k-1)\cdot(\ell+m+k+1)$. ■

Lemma 3.4

Let $G=(V,E)$ be a graph not isomorphic to a star, k be an integer such that $1 \leq k < |V|-1$ and $G'_k = (V', E')$ be the graph obtained through Construction 3.2. Let $S \subset V'$ be such that $|S| \geq |L| + |M| + k$. Then a vertex $f \in M \cap S$ is satisfied in $G'_k[S]$ if and only if $d_{\bar{S}}(f) < |\bar{S}|$.

Proof. If $d_{\bar{S}}(f) = |\bar{S}|$, f is obviously not satisfied. If $d_{\bar{S}}(f) < |\bar{S}|$, then notice that $d(f) = |L| + |N| - 2 = |V'| - |M| - 2$. Therefore, $d_S(f) = d(f) - d_{\bar{S}}(f) \geq |V'| - |M| - 2 - |\bar{S}| + 1 = |S| - |M| - 1$. Also, $|\bar{S}| \leq |N| - k$. Consequently, according to Lemma 3.3,

$$\frac{d_S(f)}{|S|-1} \geq \frac{|S|-|M|-1}{|S|-1} \geq \frac{|L|+k-1}{|L|+|M|+k-1} = \frac{|N|-k-1}{|N|-k} \geq \frac{d_{\bar{S}}(f)}{|\bar{S}|}.$$

■

Lemma 3.5

Let $G=(V,E)$ be a graph not isomorphic to a star, k be an integer such that $1 \leq k < |V|-1$, and $G'_k = (V', E')$ be the graph obtained through Construction 3.2. Let $S_1 \subset V'$ such that $G'_k[S_1]$ is a PDS and $|S_1| \geq |L| + |M| + k$. Then there exists $S_2 \subset V'$ such that $G'_k[S_2]$ is a PDS, $|S_2| \geq |S_1|$ and $L \cup M \subseteq S_2$. Moreover, S_2 can be found in polynomial time.

Proof. First, we prove that $M \subset S_1$. As $|S_1| \geq |L|+|M|+k > |M|+|N|$, then $S_1 \cap L \neq \emptyset$. Take a vertex $z \in S_1 \cap L$ and notice that since $d(z) = |M|$, then $d_{\bar{S}_1}(z) = |M \setminus S_1|$. The vertex z is satisfied in $G'_k[S_1]$ if and only if

$$\frac{|M| - d_{\bar{S}_1}(z)}{|L|+|M|+k-1} \geq \frac{d_{S_1}(z)}{|S_1|-1} \geq \frac{d_{\bar{S}_1}(z)}{|\bar{S}_1|} \geq \frac{d_{\bar{S}_1}(z)}{|N|-k}.$$

This implies that

$$\begin{aligned} & |M| \cdot (|N| - k) - d_{\bar{S}_1}(z) \cdot (|N| - k) \geq d_{\bar{S}_1}(z) \cdot (|L| + |M| + k - 1) \\ \iff & |M| \cdot (|N| - k) - d_{\bar{S}_1}(z) \cdot (|N| - k) \geq d_{\bar{S}_1}(z) \cdot |M| \cdot (|N| - k) \\ \iff & |M| \cdot (|N| - k) \geq d_{\bar{S}_1}(z) \cdot (|M| + 1) \cdot (|N| - k) \\ \iff & 0 \geq d_{\bar{S}_1}(z) \,. \end{aligned}$$

Thus, we have $d_{\bar{S}_1}(z) = 0$ and conclude that $M \subset S_1$.
Let $S_2 := S_1 \cup L \cup M$ and $f \in M$. As f is satisfied in $G'_k[S_1]$, according to Lemma 3.4, we have $d_{\bar{S}_1}(f) < |\bar{S}_1|$. Since f is connected to all the vertices in L, necessarily $d_{\bar{S}_2}(f) < |\bar{S}_2|$ and f remains satisfied in $G'_k[S_2]$. Obviously, the vertices in L are satisfied in $G'_k[S_2]$ since all their neighbours are in M. This is also true for the vertices in $N \cap S_2$. ■

Notice that Lemma 3.5 implies that there exists a PDS of maximum size that is connected in G'_k. Hence, the following result also holds when looking for a connected PDS.

Theorem 3.2

MAX PROPORTIONALLY DENSE SUBGRAPH is NP-hard on bipartite graphs.

Proof. Let $G = (V, E)$ be a graph not isomorphic to a star and $k \in \{1, \ldots, |V| - 2\}$. Notice that since G is connected and not isomorphic to a star, then there is no independent set of size $|V| - 1$ in G. Let $G'_k = (V', E')$ be the graph obtained through Construction 3.2. We claim that there is an independent set of size at least k in G if and only if there is a PDS of size at least $|L| + |M| + k$ in G'_k.
Let R be an independent set of G of size at least k. In G'_k, we define $S := L \cup M \cup R$ and $\bar{S} := V' \setminus S$. First, note that $R \subseteq N$ thus $\bar{S} = N \setminus R$. The vertices in $L \cup R$ are obviously satisfied in $G'_k[S]$ as all their neighbours are in S. Hence, if there exist vertices not satisfied in $G'_k[S]$, then they must belong to the set M. Consider a vertex $e \in M$. Since R is an independent set of G, then for each edge $e = \{u, v\} \in E$ at most one of the vertices u and v belongs to R, and, therefore, at least one belongs to $\bar{S}$. Therefore, the vertex $e \in M$ is not adjacent to at least one vertex in $\bar{S}$ and thus $d_{\bar{S}}(f) < |\bar{S}|$. According to Lemma 3.4, e is satisfied in $G[S]$. Thus $G[S]$ is a PDS of size at least $|L| + |M| + k$.

Let $S \subset V'$ be of size at least $|L| + |M| + k$ such that $G'_k[S]$ is a PDS. According to Lemma 3.5, there exists $S' \subset V'$ such that $G'_k[S']$ is a PDS, $|S'| \geq |S|$ and $L \cup M \subseteq S'$. We claim that $R' := S' \cap N$ is an independent set of G of size at least k. Obviously $|R'| \geq k$. Lemma 3.4 states that for all satisfied vertices $e \in M$, $d_{\bar{S}'}(e) < |\bar{S}'|$. Therefore, as $d_N(e) = |N| - 2$ and $\bar{S}' \subseteq N$, there is at most one vertex $u \in S' \cap N$ not adjacent to e. From Construction 3.1, if there is no edge between the vertices $e \in M$ and $u \in N$ in G'_k, then $u \in e$ in G. Hence, the edge $e \in E$ in G has at most one vertex $u \in R'$. Thus, R' is an independent set of size at least k. ■

Below, we prove that deciding if a subset of vertices can be extended into a larger subset which induces a PDS is NP-complete. We obtain as a corollary that deciding if a PDS is inclusion wise maximal is coNP-complete.

PDS Extension

Input: A graph $G = (V, E)$, $U \subset V$.
Question: Is there a vertex subset $S \subset V$ such that $U \subset S$ and $G[S]$ is a proportionally dense subgraph?

To prove that PDS Extension is NP-complete, we use again Construction 3.2.

Lemma 3.6

Let $G = (V, E)$ be a graph not isomorphic to a star, k be an integer such that $1 \leq k < |V| - 1$, and $G'_k = (V', E')$ be the graph obtained through Construction 3.2. Let $S \subset V'$ be such that $L \cup M \subset S$ and $G'_k[S]$ is a PDS. Then $|S| \geq |L| + |M| + k$.

Proof. Let $u \in S \cap N$, and notice that $d_S(u) < |M|$, so there exists a vertex in M which is not connected to u. Let $f \in M$ be such a vertex. Note that $d_S(f) \leq |S| - |M| - 1$ and $d_{\bar{S}}(f) \geq |\bar{S}| - 1$, as f is not connected to u. Let $k' := |N \setminus \bar{S}| = |N| - |\bar{S}|$. We claim that $k' \geq k$. Suppose by contradiction that $k' < k$. Then $\frac{|L|+k'-1}{|L|+|M|+k'-1} < \frac{|L|+k-1}{|L|+|M|+k-1}$ and $\frac{|N|-k-1}{|N|-k} < \frac{|N|-k'-1}{|N|-k'}$. According to Lemma 3.3, we conclude that $\frac{|L|+k'-1}{|L|+|M|+k'-1} < \frac{|N|-k'-1}{|N|-k'}$. Therefore,

$$\frac{d_S(f)}{|S|-1} \leq \frac{|L|+k'-1}{|L|+|M|+k'-1} < \frac{|N|-k'-1}{|N|-k'} \leq \frac{d_{\bar{S}}(f)}{|\bar{S}|},$$

which contradicts that f is satisfied, thus that $G'_k[S]$ is a PDS. We conclude that $|S| = |L| + |M| + k' \geq |L| + |M| + k$. ■

Theorem 3.3

PDS Extension is NP-complete on bipartite graphs.

Proof. Obviously, PDS Extension is in NP. Let $G = (V, E)$ be a graph not isomorphic to a star and $k \in \{1, \ldots, |V| - 1\}$. Notice that since G is connected and not isomorphic to a star, then there is no independent set of size $|V| - 1$ in G, so we can consider that $k \leq |V| - 2$. Let $G'_k = (V', E')$ be the graph obtained through Construction 3.2. We claim that there is an independent set of size at least k in G if and only if there is PDS of size of size at least $|L| + |M| + k$ in G'_k.

Assume there exists an independent set of size k in G. Then there exists $S \subset V'$ of size $|S| \geq |L| + |M| + k$ such that $G'_k[S]$ is a PDS, and $L \cup M \subset S$ (see proof of Theorem 3.2).

According to Lemma 3.6, if there exists $S \subset V'$ such that $G'_k[S]$ is a PDS and $L \cup M \subset S$, then $|S| \geq |L| + |M| + k$. Therefore, there exists an independent set of size at least k in G (see proof of Theorem 3.2).

We conclude that deciding if there exists $S \subset V'$ such that $L \cup M \subset S$ and $G'_k[S]$ is a PDS is NP-complete, and thus that PDS Extension is NP-complete on bipartite graphs. ■

Notice that the set $L \cup M$ is connected, thus if it can be extended into a PDS, then the PDS is connected. Hence, it is NP-complete to decide whether a vertex subset (inducing a connected subgraph) can be extended into a connected PDS. Furthermore, the set $L \cup M$ can induce a PDS or not, depending on the values of k and $|V|$. Indeed, $G'_k[L \cup M]$ is a PDS if and only if $\frac{|L|}{|L|+|M|-1} \geq \frac{|N|-2}{|N|}$, which implies $k \leq \frac{n}{2}$. Therefore, we conclude that deciding if a PDS is inclusion-wise maximal is coNP-complete.

Corollary 3.3.1

Let $G = (V, E)$ be a graph and $S \subset V$ such that $G[S]$ is a proportionally dense subgraph. Deciding if S is inclusion-wise maximal is coNP-complete on bipartite graphs.

3.4. Approximation of Maximum PDS

We show that Max PDS is polynomial-time 2-approximable, which establishes its APX-completeness. We also show that the ratio can be improved to $\frac{2\cdot(\Delta-1)+1}{\Delta}$ on connected graphs, where Δ is the maximum degree of the graph, using an upper-bound on the maximum possible size of a PDS.

Now, we design a polynomial-time algorithm that generates, given a graph $G = (V, E)$, a PDS of size at least $\lceil \frac{|V|}{2} \rceil$.

Lemma 3.7

Let $G = (V, E)$ be a graph and $S \subset V$ be a set of vertices of size $\frac{|V|}{2}$ or $\frac{|V|}{2} + 1$ for $|V|$ even, and $\frac{|V|+1}{2}$ for $|V|$ odd. If S is not a PDS in G, then there exists a vertex $u \in S$ such that $d_S(u) < d_{\bar{S}}(u)$ if $|S| \leq \frac{|V|+1}{2}$, and $d_S(u) \leq d_{\bar{S}}(u)$ otherwise.

Proof. Let $S \subset V$ be a subset such that $G[S]$ is not a PDS. Then, there exists a vertex $u \in S$ that is not satisfied in $G[S]$, and therefore $|\bar{S}| \cdot d_S(u) < (|S| - 1) \cdot d_{\bar{S}}(u)$ $(*)$.

- If $|S| = \lceil \frac{|V|}{2} \rceil$, the inequality $(*)$ implies $\lfloor \frac{|V|}{2} \rfloor \cdot d_S(u) < (\lceil \frac{|V|}{2} \rceil - 1) \cdot d_{\bar{S}}(u) \leq \lfloor \frac{|V|}{2} \rfloor \cdot d_{\bar{S}}(u)$, and hence $d_S(u) < d_{\bar{S}}(u)$.
- If $|S| = \frac{|V|}{2} + 1$ ($|V|$ even), assume by contradiction that for each vertex $v \in S$ it holds $d_S(v) > d_{\bar{S}}(v)$. In particular, the inequality $(*)$ implies $(\frac{|V|}{2} - 1) \cdot (d_{\bar{S}}(u) + 1) < \frac{|V|}{2} \cdot d_{\bar{S}}(u)$, which is true if and only if $d_{\bar{S}}(u) \geq \frac{|V|}{2}$. Thus, $d(u) = d_S(u) + d_{\bar{S}}(u) > |V| - 1$, a contradiction.

■

Theorem 3.4

For any graph $G = (V, E)$, a proportionally dense subgraph of size $\lceil \frac{|V|}{2} \rceil$ or $\lceil \frac{|V|}{2} \rceil + 1$ can be constructed in $O(|V| \cdot |E|)$ time.

Proof. First, we show that Algorithm 1 terminates and returns a PDS of size $\lceil \frac{|V|}{2} \rceil$ or $\lceil \frac{|V|}{2} \rceil + 1$.

- **Case 1: $|V|$ is odd.** Notice that at the end of each loop, the set S is modified without changing its size $|S| = \frac{|V|+1}{2} = \lceil \frac{|V|}{2} \rceil$. If $G[S]$ is not a PDS, then according to Lemma 3.7 there exists an unsatisfied

vertex $v \in S$ for which $d_S(v) < d_{\bar{S}}(v)$. Therefore, the vertex u chosen within the loop has the property $d_{\bar{S}}(u) - d_S(u) > 0$. Thus, the size of the cut between S and $\bar{S}$ decreases after each loop and the algorithm terminates.

- **Case 2: $|V|$ is even.** Notice that Algorithm 1 starts with $|S| = \frac{|V|}{2}$. If $G[S]$ is not a PDS, then due to Lemma 3.7, there exists a vertex $v \in S$ such that $d_S(v) < d_{\bar{S}}(v)$. The selection of the vertex $u \in S$ inside the loop ensures that the size of the cut between S and $\bar{S}$ strictly decreases at the end of the loop. Now, observe that after the first loop, $|S| = \frac{|V|}{2} + 1$. If $G[S]$ is not a PDS, according to Lemma 3.7, there exists a vertex $v \in S$ such that $d_S(v) \leq d_{\bar{S}}(v)$. Therefore, the vertex u inside the loop has $d_S(u) \leq d_{\bar{S}}(u)$. Obviously, after the second loop, $|S| = \frac{|V|}{2}$. Since after each loop $|S|$ alternates between $\frac{|V|}{2}$ and $\frac{|V|}{2} + 1$, the size of the cut between S and $\bar{S}$ strictly decreases every two loops, and the algorithm terminates.

It is easy to see that the while-loop is called at most $O(|E|)$ times. Now, we prove how one can obtain a $O(|V| \cdot |E|)$ running time by computing Lines 2 to 4 in $O(|V|)$ time.

Preprocessing. Once S has been defined at Line 1, compute and store the following *properties* for each vertex $u \in V$: $d_S(u)$, $d_{\bar{S}}(u)$, and whether u belongs to S or $\bar{S}$. The computation of these properties for all the vertices can be done in $O(|E|)$ time. While computing the properties, one can also choose a vertex $u \in S$ that maximises $d_{\bar{S}}(u) - d_S(u)$ (as in Line 3).

Main loop. If $d_{\bar{S}}(u) - d_S(u) > 0$, then S is not a PDS. However, if $d_{\bar{S}}(u) - d_S(u) = 0$, then S is a PDS if and only if $|S| < \frac{|V|}{2} + 1$ (so we compute Line 2 in constant time). Therefore, if S is not a PDS, set $S := \bar{S} \cup \{u\}$ (as in Line 4), update the properties of all the vertices and select $u \in S$ maximising $d_{\bar{S}}(u) - d_S(u)$ (as in Line 3) in $O(|V|)$. Then, repeat the main loop. ■

Algorithm 1 implies several consequences. Firstly, it gives a 2-approximation algorithm since any PDS has size at most $|V| - 1$. Besides, it shows that the decision version associated to Max PDS is FPT when parameterized by the natural parameter k (*i.e.* the size of the solution). Indeed, if the parameter $k \leq \lceil \frac{|V|}{2} \rceil$, then a PDS of size greater than k can be found in polynomial time using Algorithm 1. On the other hand, if $k > \lceil \frac{|V|}{2} \rceil$, then we have $|V| < 2k$ and an exhaustive research can be done in $O(2^{2k})$ operations.

Algorithm 1: Find a PDS of size $\lceil \frac{|V|}{2} \rceil$ or $\lceil \frac{|V|}{2} \rceil + 1$.

Input: $G = (V, E)$ a graph.
Output: $S \subset V$ such that $G[S]$ is a PDS.

```
1 Let S ⊂ V with |S| = ⌈|V|/2⌉;
2 while G[S] is not a PDS do
3     Let u ∈ S such that d_S̄(u) − d_S(u) is maximum;
4     S := S̄ ∪ {u};
5 return S;
```

We show in the following how the calculation of the approximation ratio can be improved on connected with regard to the maximum degree of the graph.

Lemma 3.8

Let $G = (V, E)$ be a connected graph and $S \subset V$ be a PDS in G. Then $|S| \leq \lfloor \frac{(\Delta(G)-1)\cdot|V|+1}{\Delta(G)} \rfloor$.

Proof. Let v be a vertex of S with at least one neighbour in $\bar{S} = V \setminus C$ (such vertex exists since G is connected). Since S is a PDS, v fulfils the proportion condition, that is to say: $\frac{\Delta(G)-1}{|S|-1} \geq \frac{d_S(v)}{|S|-1} \geq \frac{d_{\bar{S}}(v)}{|\bar{S}|} \geq \frac{1}{|V|-|S|}$ which implies that $|S| \leq \frac{(\Delta(G)-1)\cdot|V|+1}{\Delta(G)}$. Since $|S|$ is an integer, we obtain $|S| \leq \lfloor \frac{(\Delta(G)-1)\cdot|V|+1}{\Delta(G)} \rfloor$. ■

Proposition 3.3

Max PDS is polynomial-time $\frac{2\cdot(\Delta(G)-1)+1}{\Delta(G)}$-approximable on connected graphs.

Proof. Let $G = (V, E)$ be a connected graph, S be a solution given by Algorithm 1 and $OPT(G)$ denote the size of a PDS of maximum size in G. According to Lemma 3.8 we have $OPT(G) \leq \frac{(\Delta(G)-1)\cdot|V|+1}{\Delta(G)}$. Thus we obtain $|S| \geq \frac{|V|}{2} \geq \frac{\Delta(G)}{2\cdot(\Delta(G)-1)+1} \cdot \frac{(\Delta(G)-1)\cdot|V|+1}{\Delta(G)} \geq \frac{\Delta(G)}{2\cdot(\Delta(G)-1)+1} \cdot OPT(G)$. ■

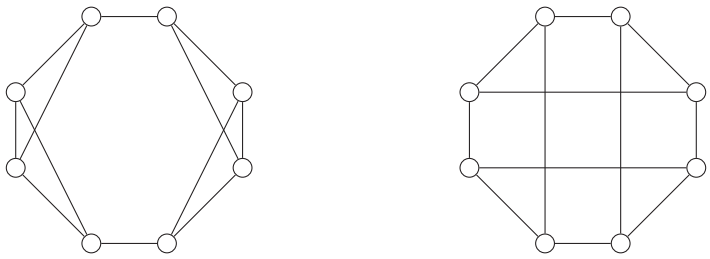
As mentioned above, regardless of the connected or the maximum degree of the graph, Algorithm 1 gives a 2-approximation for Max PDS since $|S| \geq \frac{1}{2} \cdot |V| \geq \frac{1}{2} \cdot OPT(G)$.

Theorem 3.5

MAX PDS is APX-complete.

3.5. Hamiltonian cubic graphs

In this section, we prove that all Hamiltonian cubic graphs of order n, except two graphs (see Fig. 3.4), have a proportionally dense subgraph of the maximum possible size $\lfloor \frac{2n+1}{3} \rfloor$ (see Lemma 3.8 for an upper bound on a PDS size). Furthermore, we show that such a PDS can be found in linear time if a Hamiltonian cycle is given in the input. Note that almost all cubic graphs are Hamiltonian, as proved in [78].



H_1 H_2

Figure 3.4: Two Hamiltonian cubic graphs with 8 vertices without a PDS of maximum possible size $\lfloor \frac{2\times 8+1}{3} \rfloor = 5$.

We represent a Hamiltonian cubic graph of order n as a cycle with the vertices labelled in such a way that $(0, 1, \ldots, n-1)$ is a Hamiltonian cycle and a set of edges between non-successive vertices in the Hamiltonian cycle. We always refer to this cycle when we say *the* Hamiltonian cycle of a graph. To avoid tedious notations, we use $i \in \mathbb{N}$ (with $0 \in \mathbb{N}$) to refer to the vertex labelled by i mod n.

Definition 3.2

Let $G = (V, E)$ be a Hamiltonian cubic graph, $u \in V$. Let P be a set of successive vertices in the Hamiltonian cycle labelled with u, $u+1$, …, $u-k-1$, with $k \geq 3$. The set P is called a *shift* if the first and the last vertices of the sequence, u and $u-k-1$, are such that $d_P(u) = d_P(u-k-1) = 2$.

Notice that a shift P contains $|V| - k$ vertices. Also, any vertex of P has at least two neighbours in P. Consequently, if $k \geq \lceil \frac{|V|-1}{3} \rceil$, then $|P| \leq \lfloor \frac{2\cdot|V|+1}{3} \rfloor$, and the following holds for any $u \in P$:

$$\frac{d_P(u)}{|P|-1} \geq \frac{2}{|V|-k-1} \geq \frac{1}{k} \geq \frac{d_{\bar{P}}(u)}{|\bar{P}|}.$$

Thus, $G[P]$ is a PDS. If $k = \lceil \frac{|V|-1}{3} \rceil$, then $G[P]$ is a PDS of the maximum possible size $\lfloor \frac{2\cdot|V|+1}{3} \rfloor$ (see Lemma 3.8) and we call P a *good shift*. On the other hand, if $k = \lceil \frac{|V|-1}{3} \rceil - 1$, then the size of P is one vertex larger than the size of the maximum possible PDS, and thus $G[P]$ is not a PDS. Such a shift is called an *almost good shift*.

In the following, we prove that either G contains a good shift or we can find an almost good shift P and a vertex $v \in P$ such that $G[P \setminus \{v\}]$ is a proportionally dense subgraph of the maximum possible size $\lfloor \frac{2\cdot|V|+1}{3} \rfloor$.

Definition 3.3

Let $G = (V, E)$ be a Hamiltonian cubic graph. For each $v \in V$, we denote by $c(v)$ the non-successive neighbour of v in the Hamiltonian cycle. Additionally, we define the subsets of vertices L and R in the following way for $k := \lceil \frac{|V|-1}{3} \rceil$:

- $L := \{u \in V : c(u) \in \{u-k, u-k+1, , \ldots, u-2\}\}$;
- $R := \{u \in V : c(u) \in \{u+2, u+3, \ldots, u+k\}\}$.

For a Hamiltonian cubic graph $G = (V, E)$ and $u \in V$, notice that $u \in L$ if and only if $c(u) \in R$, and symmetrically $u \in R$ if and only if $c(u) \in L$. This particularly implies that $|L| = |R| \leq \frac{|V|}{2}$. Moreover, notice that for a vertex $u \in L$, the set $P := \{u, u+1, \ldots, u-k-1\}$ cannot be a good shift, since $d_P(u) = 1$. In the same way, if $u \in R$, the set $P := \{u+k+1, u+k+2, \ldots, u-1, u\}$ cannot be a good shift, since $d_P(u) = 1$. These observations are summed up in the following lemma.

Lemma 3.9

Let $G = (V, E)$ be a Hamiltonian cubic graph, $k := \lceil \frac{|V|-1}{3} \rceil$ and $u \in V$. If $u \notin L$ and $(u-(k+1)) \notin R$, then the set $\{u, u+1, \ldots, u-(k+1)-1, u-(k+1)\}$ is a good shift. Symmetrically, if $u \notin R$ and $(u+k+1) \notin L$, then the set $\{u+k+1, u+k+2, \ldots, u-1, u\}$ is a good shift.

Proof. The proof is straightforward. Since $u \notin L$ and $(u-(k+1)) \notin R$, we have $d_P(u) = d_P(u-(k+1)) = 2$, where $P := \{u, u+1, \ldots, u-(k+1)\}$. The other case is similar. ■

An important consequence of Lemma 3.9 is that if G is a Hamiltonian cubic graph with no good shift, then we can define subsets of vertices that must be either in L or in R. To define such subsets we introduce the following notation.

Definition 3.4

Let $G = (V, E)$ be a Hamiltonian cubic graph and $u \in V$. We define the vertex subset $<u>:= \{v \in V : v \equiv u \pmod{(k+1)}\}$ where $k := \lceil \frac{|V|-1}{3} \rceil$.

Corollary 3.5.1

Let $G = (V, E)$ be a Hamiltonian cubic graph with no good shift and $u \in V$:

- if $u \notin R$ then $<u> \subseteq L$,
- if $u \notin L$, then $<u> \subseteq R$,
- $|L| = |R| = \frac{|V|}{2}$.

Proof. First notice that for any integer $\delta \geq 1$, $u - \delta \cdot (k+1) \equiv u - \delta \cdot (k+1) + |V| \cdot \delta \cdot (k+1) \pmod{|V|} \equiv u + \delta \cdot (|V|-1) \cdot (k+1) \pmod{|V|}$. Moreover, $u \equiv u + |V| \cdot (k+1) \pmod{|V|}$. Thus, we have $\{u - \delta \cdot (k+1) : \delta \geq 1, \delta \in \mathbb{N}\} = \{u + \delta \cdot (k+1) : \delta \geq 1, \delta \in \mathbb{N}\} = <u>$.
Now, if $u \notin R$, then, with our assumption that G has no good shift and Lemma 3.9, we derive that $<u> = \{u + \delta \cdot (k+1) : \delta \geq 1, \delta \in \mathbb{N}\} \subseteq L$. Symmetrically, if $u \notin L$, then $\{u - \delta \cdot (k+1) : \delta \geq 1, \delta \in \mathbb{N}\} \subseteq R$.
This implies that for any vertex $u \in V$, either $u \in L$ or $u \in R$. Finally, since $u \in L$ if and only if $c(u) \in R$ and $u \in R$ if and only if $c(u) \in L$, then it is obvious that $|L| = |R| = \frac{|V|}{2}$. ■

Let $G = (V, E)$ be a Hamiltonian cubic graph with no good shift and $d := \gcd(k+1, |V|)$, where $\gcd(k+1, |V|)$ is the greatest common divisor of $(k+1)$ and $|V|$. We show that V can be partitioned into d subsets of vertices $<0>, <1>, \ldots, <d-1>$. This partition will be useful to find an *almost good shift* P and a vertex to remove from P in order to obtain a PDS in G. This result comes from a basic property of the cyclic group $\mathbb{Z}/n\mathbb{Z}$ that we recall in the following lemma.

Lemma 3.10

Let $\alpha \geq 1$ and $\beta \geq 1$ be positive integers, and $d := \gcd(\alpha, \beta)$. If all integers are considered mod α, then $\{0, 1, \ldots, \alpha - 1\} = \cup_{i \in \{0,1,\ldots,d-1\}} < i >$ where $< i >:= \{l : l \equiv i \pmod{\beta} \text{ and } l \in \{0, 1, \ldots, \alpha - 1\}\}$. Moreover, for any $i, j \in \{0, 1, \ldots, d - 1\}$ with $i \neq j$, $< i > \cap < j >= \emptyset$.

Proof. First, we prove that for any $u \geq d$, $u \in < i >$ for some $i \in \{0, 1, \cdots, d - 1\}$. Let $u \geq d$. Then there exist two integers a, b with $b \leq d-1$, such that $u = a \cdot d + b$. Moreover, there exist two integers c, f such that $c \cdot \beta + f \cdot \alpha = d$ since $d = \gcd(\alpha, \beta)$. Then, $u = a \cdot c \cdot \beta + a \cdot f \cdot \alpha + b \equiv b + a \cdot c \cdot \beta \pmod{\alpha}$. Thus, $u \in < b >$ with $b \leq d-1$. This proves that any integer is in a set $< i >$ for some $i \leq d-1$, *i.e.* $\{0, 1, \cdots, \alpha-1\} = \cup_{i \in \{0,1,\cdots,d-1\}} < i >$. To prove the second part of the statement, we first show that $\alpha = | < u > | \cdot d$ for any $u \in \{0, 1, \cdots, d - 1\}$. Let $u \in \{0, 1, \cdots, d - 1\}$ and $p \geq 1$ be the smallest integer such that $u + p \cdot \beta \equiv u \pmod{\alpha}$. Notice that $| < u > | = p$ and let us show that $\alpha = p \cdot d$. Let α', β' be two integers such that $\alpha = \alpha' \cdot d$, $\beta = \beta' \cdot d$ and $\gcd(\alpha', \beta') = 1$. We prove that $\alpha' = p$ by verifying that α' divides p and p divides α'. First, notice that $u + \alpha' \cdot \beta = u + \alpha' \cdot k' \cdot d = u + \alpha \cdot \beta' \equiv u \pmod{\alpha}$. Thus, p divides α'. On the other hand, recall that $u + p \cdot \beta \equiv u \pmod{\alpha}$ and notice that $u + p \cdot \beta = u + p \cdot \beta' \cdot d$, then $p \cdot \beta' \cdot d \equiv 0 \pmod{\alpha}$. This implies that α divides $p \cdot \beta' \cdot d$, and thus α' divides $p \cdot \beta'$. Since $\gcd(\alpha', \beta') = 1$, α' divides p. Now, notice that two sets $< i >, < j >$ for some integers i, j are either equal or disjoint. Since for any $u \in \{0, 1, \cdots, \alpha - 1\}$ we have $| < u > | = \frac{\alpha}{d}$, then obviously all sets $< i >$, $i \in \{0, 1, \cdots, d - 1\}$ are disjoint. ■

In the following lemma, we summarise the possible values of $\gcd(n, k + 1)$ for some specific values of n and k.

Lemma 3.11

Let n be an even integer, $n \geq 4$. Then:

- if $n = 3k - 1$, then $\gcd(n, k + 1) \in \{2, 4\}$,
- if $n = 3k$, then $\gcd(n, k + 1) \in \{1, 3\}$,
- if $n = 3k + 1$, then $\gcd(n, k + 1) = 2$.

Proof. Consider the case $n = 3k - 1$, then $d := \gcd(k+1, 3k-1) = \gcd(k+1, 3k-1-2(k+1)) = \gcd(k+1, k-3) = \gcd(4, k-3)$. As n is even, then k is odd and $d \in \{2, 4\}$. The other cases can be proved using the same reasoning. ■

Firstly, we show that if $|V| = 3k$, then there is always a good shift in G.

Corollary 3.5.2

Let G be a Hamiltonian cubic graph with $3k$ vertices, $k \geq 2$. Then G has a good shift.

Proof. Suppose by contradiction that there is no good shift in $G = (V, E)$. Notice that if $|V| = 3k$, then $k = \lceil \frac{|V|-1}{3} \rceil$. Let $d := \gcd(k+1, |V|)$. From Lemma 3.11 we get $d \in \{1, 3\}$. According to Corollary 3.5.1, $|L| = |R| = \frac{|V|}{2}$. If $d = 1$, then $V = <0>$ (Lemma 3.10), and hence $V = L$ or $V = R$, which is impossible. If $d = 3$, then $|V| = <0> \cup <1> \cup <2>$ (Lemma 3.10). According to Corollary 3.5.1, $<i> \subseteq L$ or $<i> \subseteq R$ for any $i \in \{0, 1, 2\}$, thus $|R| \neq |L|$, which is not possible. ■

From Lemma 3.10 and Lemma 3.11, if a Hamiltonian cubic graph $G = (V, E)$ has no good shift, then V can be written as $V = <0> \cup <1> \cup <2> \cup <3>$ (we may have $<0> = <2>$ and $<1> = <3>$). Hence those graphs can be split into two categories:

- *type RLRL:* for any vertices $i, i+1$ with $i \in V$, we have $i \in L$ and $i+1 \in R$, or $i \in R$ and $i+1 \in L$. In this case, we always assume without loss of generality that $R = <0> \cup <2>$ and $L = <1> \cup <3>$.
- *type RRLL:* there exist two vertices $i, i+1$ with $i \in V$ such that $i, i+1 \in L$ or $i, i+1 \in R$. In this case, we always assume without loss of generality that $R = <0> \cup <1>$ and $L = <2> \cup <3>$.

Now, we show that if a Hamiltonian cubic graph G has no good shift, then there exists an almost good shift P in G (Lemma 3.12) and a vertex $v \in P$ such that $G[P \setminus \{v\}]$ is a PDS (Lemma 3.13 and Theorem 3.6).

Lemma 3.12

Any Hamiltonian cubic graph with no good shift has an almost good shift.

Proof. Let $G = (V, E)$ be a Hamiltonian cubic graph with no good shift, $k = \lceil \frac{|V|-1}{3} \rceil$ and $d := \gcd(k+1, |V|)$. Since G has no good shift, according to Lemma 3.11 and Corollary 3.5.2, $d \in \{2, 4\}$ and $|V| = 3k-1$ or $|V| = 3k+1$. From Corollary 3.5.1, we know that each vertex in V belongs to either L or R.

- Case 1: *G is of type RLRL.* Let $P := \{0, 1, \cdots, -k\}$. Since $|V|$ is even, then $|P|$ is even. Therefore, since two vertices $i, i+1 \in P$ do not both belong to L or R, then the vertex $-k$ belongs to L. Then the set P fulfils the requirements.
- Case 2: *G is of type RRLL.* Consider the set $P := \{1, 2, \cdots, -k+1\}$. According to Lemma 3.11, since $d = 4$, $|V| = 3k-1$. Hence, $-k+1 = 2-(k+1) \in <2>$. Thus $-k+1 \in L$ and P fulfils the requirements.

■

Recall that the graphs H_1 and H_2 from Fig. 3.4 have no proportionally dense subgraph of the maximum possible size. In Theorem 3.6, we show that these are the only cubic Hamiltonian graphs with this property.

Before proving the main theorem, we first deal with small graphs ($|V| < 20$) that are particular cases that need to be treated independently.

Lemma 3.13

Let $G = (V, E)$ be a Hamiltonian cubic graph not isomorphic to H_1 or H_2 with $|V| < 20$. Then there exists a PDS of size $\lfloor \frac{2 \cdot |V|+1}{3} \rfloor$ in G.

Proof. Let $k = \lceil \frac{|V|-1}{3} \rceil$. Since G is cubic, its number of vertices is even. From Lemma 3.11, $\gcd(k+1, |V|) \in \{1, 2, 3, 4\}$. If $\gcd(k+1, |V|) \in \{1, 3\}$, then there exists a good shift from Corollary 3.5.2. We suppose then that $\gcd(k+1, |V|) \in \{2, 4\}$. The following cases remain:

- If $|V| = 4$, then G is the complete graph K_4, and any set of 3 vertices induces a PDS of size $\lfloor \frac{2 \cdot 4+1}{3} \rfloor$.
- If $|V| = 8$, we claim that G must have a good shift. By contradiction, suppose that G has no good shift. If G is of *type RRLL* then G is isomorphic to H_1, and if G is of *type RLRL* then G is isomorphic to H_2, which is impossible since we assumed that G is not isomorphic to H_1 or H_2.

- If $|V| = 10$ and G has no good shift, since $\gcd(k+1, |V|) = 2$, G is necessarily of *type RLRL* and $c(0) = 3$, $c(1) = 8$, $c(2) = 5$, $c(4) = 7$, $c(6) = 9$. In this case, $V \setminus \{0, 6, 9\}$ induces a PDS of size $\lfloor \frac{2\cdot 10+1}{3} \rfloor$.
- If $|V| = 14$, if G has no good shift, since $\gcd(k+1, |V|) = 2$, then G is necessarily of *type RLRL*. Following Lemma 3.12, let $P := \{0, 1, \cdots, 9\}$ be an almost good shift and:
  - If $c(6) \neq 9$, notice that $c(7), c(5) \in P$ (since $5, 7 \in L$) and $c(6) \in V \setminus P$. Thus $G[P \setminus \{6\}]$ is a PDS of size $\lfloor \frac{2\cdot 14+1}{3} \rfloor$. If $c(3) \neq 0$, the case is symmetrical.
  - If $c(3) = 0$ and $c(6) = 9$, notice that $c(3) \in P$, $c(5) \in P$ and $d_P(c(4)) = 3$ since $c(4) \neq 9$). Thus $G[P \setminus \{4\}]$ is a PDS of size $\lfloor \frac{2\cdot 14+1}{3} \rfloor$.
- If $|V| = 16$, if G has no good shift, since $\gcd(k+1, |V|) = 2$, G is necessarily of *type RLRL*. Following Lemma 3.12, let $P := (0, 1, \cdots, -k)$ be an almost good shift. Since $0 \in R$, we have either $c(0) = 3$ or $c(0) = 5$. In each case, the graph is completely determined due to the constraints. In the first case, $P \setminus \{4\}$ induces a PDS of size $\lfloor \frac{2\cdot 16+1}{3} \rfloor$. In the second case, $P \setminus \{3\}$ induces a PDS of size $\lfloor \frac{2\cdot 16+1}{3} \rfloor$.

In each case, if G is not isomorphic to H_1 or H_2, then either G has a good shift which is a PDS of size $\lfloor \frac{2\cdot |V|+1}{3} \rfloor$, or we give a PDS of such size. ■

Theorem 3.6

Let $G = (V, E)$ be a Hamiltonian cubic graph not isomorphic to H_1 or H_2. Then there exists a connected PDS of size $\lfloor \frac{2\cdot |V|+1}{3} \rfloor$ in G.

Proof. If $|V| < 20$, then there is a PDS of size $\lfloor \frac{2\cdot |V|+1}{3} \rfloor$ in G from Lemma 3.13. Now we suppose that $|V| \geq 20$, which implies that $k := \lceil \frac{|V|-1}{3} \rceil \geq 7$.

From Lemma 3.11, $\gcd(k+1, |V|) \in \{1, 2, 3, 4\}$. If $\gcd(k+1, |V|) \in \{1, 3\}$, then there exists a good shift (Corollary 3.5.2).

We suppose that $\gcd(k+1, |V|) \in \{2, 4\}$. If G contains a good shift, then the proof is done. Notice that in such case, the PDS is obviously connected.

Now, we assume that G has no good shift. We prove that given an almost

good shift P, there exists a vertex $u^* \in P$ such that $G[P \setminus \{u^*\}]$ is a PDS. Observe that such vertex u^* exists if and only if $c(u^*-1), c(u^*+1) \in P$, and either $c(u^*) \in V \setminus P$ or $d_P(c(u^*)) = 3$.

- If G is of *type RLRL*, then $R = <0> \cup <2>$ and $L = <1> \cup <3>$. According to Lemma 3.12, the set $P := \{0, 1, 2, \cdots, -k\}$ is an almost good shift and $0 \in R, 1 \in L$. Since $2 \in R$ and $4 \in R$, then $c(2) \in P$ and $c(4) \in P$. If $c(3) \neq 0$, then $c(3) \in V \setminus P$ since $3 \in L$. Thus, $G[P \setminus \{3\}]$ is a PDS of size $\lfloor \frac{2\cdot|V|+1}{3} \rfloor$. Symmetrically, if $c(-k-3) \neq -k$, then $c(-k-3) \in V \setminus P$ since $3 \in R$. Thus, $G[P \setminus \{-k-3\}]$ is a PDS of size $\lfloor \frac{2\cdot|V|+1}{3} \rfloor$. On the other hand, if $c(3) = 0$ and $c(-k-3) = -k$, then $c(k-1) \neq -k$ and $c(k-1) \in P$. Moreover, since $k-3 \in R$ then $c(k-3) \in P$. Therefore, $c(k-2) \in V \setminus P$ or $d_P(c(k-2)) = 3$ (since $k \geq 7$, $k-2 \neq 3$ and $c(k-2) \neq 0$). Thus, $G[P \setminus \{k-2\}]$ is a PDS of size $\lfloor \frac{2\cdot|V|+1}{3} \rfloor$. Notice that the resulting PDS is connected. Indeed, let v be the vertex we removed from the path $\{0, 1, \cdots, -k\}$. It is easy to see that, either $c(v-1) \in \{v+1, v+2, \cdots, -k\}$, or $c(v+1) \in \{0, 1, \cdots, v-1\}$ since the graph is of type $RLRL$, and thus the PDS is connected.

- If G is of *type RRLL*, then $R = <0> \cup <1>$ and $L = <2> \cup <3>$. According to Lemma 3.12, the set $P := \{1, 2, \cdots, -k+1\}$ is an almost good shift and $1 \in R, 2 \in L, -k \in R, -k+1 \in L$. Since $k+1 \in <0>$ and $k+2 \in <1>$, we necessarily have $k-1, k \in L$ and $k+1, k+2 \in R$. In this case, notice that since $k \geq 7$, $\{k-3, k-2, k-1\} \in P$. Moreover, $k-3, k-2 \in R$, which implies $c(k-3), c(k-2) \in P$. We show that either $c(k-1) \in P$ or $c(k) \in P$. Suppose that $c(k) \notin P$. Then since $k \in L$, we have $c(k) = 0$. Since $k-1 \in L$, we have $c(k-1) \in \{-1, 0, 1, \cdots, k-3\}$. Since $0 = c(k)$ and $-1 \in L$, then $c(k-1) \neq -1$ and $c(k-1) \neq 0$. Thus $c(k-1) \in \{1, 2, ..., k-3\} \subset P$. Thus either $c(k-1) \in P$ or $c(k) \in P$. Now, if $c(k-1) \in P$, then since $c(k-3) \in P$, the set $G[P \setminus \{k-2\}]$ is a PDS of size $\lfloor \frac{2\cdot|V|+1}{3} \rfloor$. Else, $c(k) \in P$ and then since $c(k-2) \in P$, the set $G[P \setminus \{k-1\}]$ is a PDS of size $\lfloor \frac{2\cdot|V|+1}{3} \rfloor$. Notice that the resulting PDS is connected. Indeed, let v be the vertex we removed from the almost good path $\{1, 2, \cdots, -k+1\}$. Again, it is easy to verify that either $v = k-2$, and then $c(k-3) \in \{k-1, k, \cdots, -k+1\}$, or $v = k-1$, and then $c(k) \in \{1, 2, \cdots, k-2\}$ since the graph is of type $RRLL$. Thus the PDS is connected.

■

According to Lemma 3.8, a PDS in a cubic graph of order n contains at most $\lfloor \frac{2n+1}{3} \rfloor$ vertices. Thus, we obtain the following corollary.

Corollary 3.6.1

Let G be a Hamiltonian cubic graph with a given Hamiltonian cycle. Then a connected proportional dense subgraph of maximum size in G can be found in linear time.

3.6. Conclusion and open problems

We prove that Max Proportionally Dense Subgraph is APX-hard even on split graphs, and NP-hard on bipartite graphs, whether the PDS is required to be connected or not. Furthermore, the problem is proved to be $\frac{2(\Delta-1)+1}{\Delta}$)-approximable, where Δ is the maximum degree of the graph. We also show that deciding if a PDS is inclusion-wise maximal is coNP-complete, even on bipartite graphs. Nonetheless, Max PDS can be solved in linear time on Hamiltonian cubic graphs if a Hamiltonian cycle is given.

However, the complexity of finding a PDS of maximum size in cubic graphs remains unknown. More specifically, the question whether a PDS of size $\lfloor \frac{2n+1}{3} \rfloor$ always exists in a cubic graph is still open (except for the two graphs given in Fig. 3.4). Also, Algorithm 1 returns, for any graph of order n, a PDS of size $\lceil \frac{n}{2} \rceil$ or $\lceil \frac{n}{2} \rceil + 1$ (in linear time), but the PDS may not be connected. An interesting open question is whether there is always a connected PDS of size at least $\lceil \frac{n}{2} \rceil$. Finally, the parameterized complexity of finding a PDS of size at least $\lceil \frac{n}{2} \rceil + k$ is unknown, where k is the size of the PDS.

Chapter 4

Colourful Components Problems

Outline

In this chapter, we consider two problems of partition of vertex-coloured graphs. We are interested in partitions into *colourful components*, that is, connected components with no two vertices of the same colour. The goal is to find such a partition minimising the number of edges with endpoints in different colourful components (Colourful Components) or minimising the number of colourful components (Colourful Partition). We prove that both problems are NP-complete on binary 4-caterpillars, on ternary 3-caterpillars and on quaternary 2-caterpillars. On the positive side, we give a linear time algorithm for 1-caterpillars with unbounded degree, even if the backbone is a cycle, which outperforms the previous best complexity on paths and can be applied to a much wider class of graphs. Finally, we show that Colourful Components remains NP-complete on 5-coloured planar graphs with maximum degree 4 and on 12-coloured planar graphs with maximum degree 3.

Some of the results presented in this chapter appear in the following paper:

- J. Chlebíková and C. Dallard, 'Towards a complexity dichotomy for colourful components problems on k-caterpillars and small-degree planar graphs', in *International Workshop on Combinatorial Algorithms*, Springer, 2019, pp. 136–147. DOI: 10.1007/978-3-030-25005-8_12.

A journal version containing our latest results is under construction.

4.1. Introduction

A *vertex coloured graph*, or simply a *coloured graph*, is a graph whose vertices are (not necessarily properly) coloured. A connected component of a coloured graph is a *colourful component* if all its vertices have different colours. A graph is said to be colourful if all its connected components are colourful.

In this chapter we focus on two closely related problems where a coloured graph and a positive integer p are given as inputs: the Colourful Components problem asks if there exist at most p edges whose removal makes the graph colourful; the Colourful Partition problem is to decide if there exists a partition of the vertex set with at most p parts such that each part induces a colourful component in the graph.

One key problem in comparative genomics is to partition a set of genes into orthologous genes, which are sets of genes in different species that have evolved through speciation events only, *i.e.* originated by vertical descent from a single gene in the last common ancestor. The problem has been modelled as a graph problem where orthologous genes translate into colourful components in the graph [1, 88]. The vertices of the graph represent the genes, and a colour is given to each vertex to symbolise the species the corresponding gene belongs to. An edge between two vertices is present in the graph if the two corresponding genes are (sufficiently) similar. The quality of a partition of a set of genes into orthologous genes can be expressed in different ways. Minimising the number of similar genes in different subsets of the partition is a well studied variant [17, 18, 53, 71, 88], and it corresponds to minimising the number of edges between the colourful components (as in Colourful Components). Alternatively, one can ask for a partition of minimum size, *i.e* which contains the minimum number of orthologous genes, or equivalently the minimum number of colourful components [1, 18, 36] (as in Colourful Partition). Another variant, not studied in this chapter, considers the objective function that maximises the number of edges in the transitive closure [1, 36, 71].

Now, we give the formal definitions of the problems considered herein.

Colourful Components

Input: A vertex-coloured graph $G = (V, E)$, a positive integer p.
Question: Are there at most p edges in E whose removal makes G colourful?

Colourful Partition

Input: A vertex-coloured graph $G = (V, E)$, a positive integer p.
Question: Is there a partition of V with at most p parts s.t. each part induces a colourful component in G?

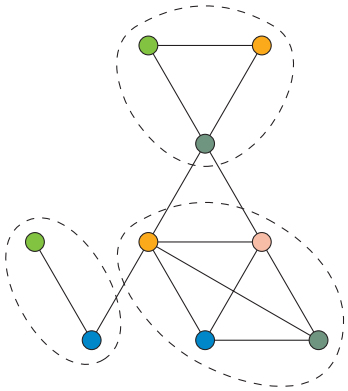


Figure 4.1: A colourful graph with outlined colourful components.

See Fig. 4.1 for an example of a graph partitioned into colourful components.

It is interesting to notice the similarities between Colourful Components and the Multicut [16, 67] and Multi-Multiway Cut [9] problems. In the Multicut problem, a graph and a set of pairs of vertices are given and the goal is to minimise the number of edges to remove in order to disconnect each pair of vertices. In the Multi-Multiway Cut problem, a graph and sets of vertices are given and the goal is to minimise the number of edges to remove in order to disconnect all paths between vertices from the same vertex set. Thus, Colourful Components is a special case where the sets of vertices form a partition.

Both Colourful Components and Colourful Partition problems can be compared to the Graph Motif problem [43]. This problem takes a coloured graph and a multiset of colours M (the motif) as input, and the goal is to determine whether there exists a connected subgraph S such that the multiset of colours used by the vertices in S corresponds exactly to M. If M is a set (where each colour appears at most once), then M is said to be colourful.

We assume that a coloured graph $G = (V, E)$ is always associated with a colouring function c from V to a set of colours, hence for each vertex $u \in V$, $c(u)$ is the colour of the vertex u. The *colour multiplicity* of G corresponds to the maximum number of occurrences of any colour in the graph. If G contains at most ℓ colours we say that G is an ℓ-coloured graph. A path P in G between two vertices u and v is called a *bad path* if $c(u) = c(v) = \gamma$ and u, v are the only two

vertices of colour γ in P. Hence, a connected component is colourful if and only if it does not contain a bad path. Lastly, given a set of edges $S \subseteq E$, we denote by $G - S$ the graph $(V, E \setminus S)$.

A *k-caterpillar*, also commonly called a *caterpillar with hairs of length at most k* [56], is a tree in which all the vertices are within distance k of a central path, called the *backbone*. Note that 2-caterpillars are also known as *lobster graphs*.

Observe that, on a tree, there is a solution to Colourful Components with p edges if and only if there is a solution to Colourful Partition with $p + 1$ parts. However, this is not the case on general graphs [18]. Both problems are known to be NP-complete on subdivided stars [36], trees of diameter at most 4 [17], and trees with maximum degree 6 [18]. Trees of diameter at most 4 are in fact a subclass of 2-caterpillars, so both problems are NP-complete on 2-caterpillars when the maximum degree is unbounded.

Overview of the results.
In Section 4.2.1, we prove that Colourful Components and Colourful Partition are NP-complete on binary 4-caterpillars and on ternary 3-caterpillar, hence with the maximum degree at most 3 or 4. This answers an open question, proposed in [18], regarding the complexity of the problems on trees with maximum degree at most 5. Nonetheless, we propose a linear time algorithm for Colourful Components and Colourful Partition on 1-caterpillars with unbounded degree in Section 4.2.2, even if the backbone induces a cycle. This result improves the complexity of the previously known quadratic-time algorithm on paths [36] and applies to a wider class of graphs. We, therefore, obtain a partial complexity dichotomy for the problems on k-caterpillars with regard to k and the maximum degree in the graph. We also consider the complexity of Colourful Components in planar graphs with small degree. It is known that the problem is NP-complete on 3-coloured graphs with maximum degree 6 [17], while Colourful Partition is NP-complete on 3-coloured 2-connected planar graphs with maximum degree 3 [18]. However, it was an open question whether Colourful Components is NP-complete on ℓ-coloured graphs with maximum degree at most 5. In Section 4.3, we answer that question and show that Colourful Components is NP-complete on 5-coloured planar graphs with maximum degree 4 and on 12-coloured planar graphs with maximum degree 3. As Colourful Components is polynomial-time solvable on graphs with maximum degree 2, our result is the best possible with regard to the maximum degree.

4.2. Complexity on k-caterpillars

In this section, we focus on the complexity of Colourful Components and Colourful Partition on k-caterpillars, depending on the value of k and the maximum degree of the graphs.

4.2.1. NP-complete cases

First, we show that Colourful Components and Colourful Partition are NP-complete on binary 4-caterpillars and ternary 3-caterpillars. We recall that a binary tree (resp. ternary) is a rooted tree in which each vertex has no more than two children (resp. three children). We propose a reduction from 3-SAT with at most three occurrence of each variable, denoted by 3, 3-SAT, which is proved NP-complete in [27].

3, 3-SAT

Input: A 3-CNF formula ϕ in which each variable occurs at most three times.

Question: Is there a satisfying assignment of ϕ?

If a clause contains a unique literal, we can set the value of the corresponding variable deterministically and reduce the number of clauses or prove that the formula is unsatisfiable. Hence, we assume that each clause of a 3-CNF formula contains at least 2 literals. Also, note that, since a variable appears at most three times, we can assume that each literal appears at least once and at most two times (otherwise we can simplify the formula by setting the variable to a value that satisfies all the clauses containing it).

Construction 4.1

Let ϕ be an instance of 3, 3-SAT, that is, a set of m clauses $C_1, C_2, \ldots, C_m$ on n variables $x_1, x_2, \ldots, x_n$, where each clause contains at most three literals and where each variable appears at most three times. We denote by m_3 the number of clauses containing three literals and by m_2 the number of clauses containing two literals.

We construct a tree T in which every variable and clause is represented by a unique gadget.

For each variable x_i, we construct a *variable gadget:* Firstly, create three vertices labelled r_{x_i}, x_i, $\bar{x}_i$, and connect x_i and $\bar{x}_i$ to r_{x_i}. If a clause C_j contains the literal x_i, then create a vertex labelled $x_{i,j}$ and connect it to the vertex x_i. Similarly, if a clause C_j contains the literal $\bar{x}_i$, then create a vertex labelled $\bar{x}_{i,j}$ and connect it to the vertex $\bar{x}_i$. Since each variable appears at most three times, then each literal appears at most two times, r_{x_i} is the root of a binary tree of depth 2. Also, since each literal appears at least once, each leaf of the gadget corresponds to one literal in a clause (see Fig. 4.2d).
For each clause C_j, we construct a *clause gadget.* Depending on the properties we want on the final tree, we propose different kinds of clause gadgets. First, suppose that C_j contains three literals ℓ_1, ℓ_2, ℓ_3:

- *Clause gadget of type A:* Create four vertices y_j, y'_j, z_j, z'_j, three vertices labelled $\ell_{1,j}$, $\ell_{2,j}$, $\ell_{3,j}$ representing the literals in C_j, and one extra vertex r_{C_j}. Then, add the edges $\{\ell_{1,j}, y_j\}$, $\{\ell_{2,j}, y'_j\}$, $\{\ell_{3,j}, z'_j\}$, $\{y_j, z_j\}$, $\{y'_j, z_j\}$, and the edges $\{z_j, r_{C_j}\}$, $\{z'_j, r_{C_j}\}$. The gadget is a binary tree of depth 3 rooted in r_{C_j} (see Fig. 4.2a).

- *Clause gadget of type B:* Create three vertices z_j, z'_j, z''_j, three vertices labelled $\ell_{1,j}$, $\ell_{2,j}$, $\ell_{3,j}$ representing the literals in C_j, and one extra vertex r_{C_j}. Then, connect z_j, z''_j and z''_j to r_{C_j}, $\ell_{1,j}$ to z_j, $\ell_{2,j}$ to z'_j and $\ell_{3,j}$ to z''_j. The gadget is a ternary tree of depth 2 rooted in r_{C_j} (see Fig. 4.2b).

Now, if C_j contains two literals $\ell_{1,j}$ and $\ell_{2,j}$:

- *Clause gadget of type C:* Create two vertices z_j, z'_j, two vertices labelled $\ell_{1,j}$, $\ell_{2,j}$ representing the literals in C_j, and one extra vertex r_{C_j}. Then, connect z_j and z'_j to r_{C_j}, $\ell_{1,j}$ to z_j and $\ell_{2,j}$ to z'_j. The gadget is a binary tree of depth 2 rooted in r_{C_j} (see Fig. 4.2c).

We now explain which clause gadgets must be used and how variable and clause gadgets must be connected together so that the final tree T has different properties. First and foremost, create a variable gadget for each variable. Also, for each clause containing two literals only, create a clause gadget of type C. Then, apply one of the following options:

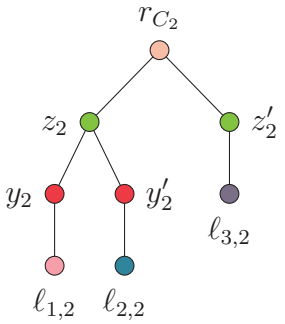
- *To get T as a binary 4-caterpillar:* Create a clause gadget of type A for each clause containing three literals, create a central path with $n + m_3 + m_2$ new vertices, and connect all r_{x_i} and r_{C_j} vertices to a different vertex of the central path.

- *To get T as a ternary 3-caterpillar:* Create a clause gadget of type A for each clause containing three literals and connect all r_{x_i} and r_{C_j} vertices together to create a central path. Alternatively, create a clause gadget of type B for each clause containing three literals, create a central path with $n + m_3 + m_2$ new vertices and connect all r_{x_i} and r_{C_j} vertices to a different vertex of the central path.

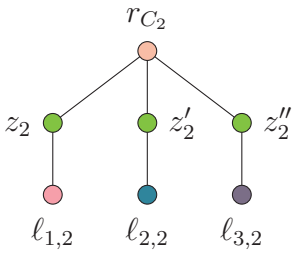
- *To get T as a quaternary 2-caterpillar:* Create a clause gadget of type B for each clause containing three literals and connect all r_{x_i} and r_{C_j} vertices together to create a central path.

In all cases, the central path corresponds to the backbone of T. We set the root r of T such that it belongs to the backbone and has minimum degree, hence two, three or four children, if T is a binary, ternary or quaternary caterpillar, respectively.
Finally, we assign a colour to each vertex in T. For each variable gadget of a variable x_i, let $c(x_i) = c(\bar{x}_i)$ be a new colour. Also, for each vertex $\widetilde{x}_{i,j} \in \{x_{i,j}, \bar{x}_{i,j}\}$, let $c(\widetilde{x}_{i,j})$ be a new colour. Then, for each clause gadget of clause C_j, if the literal $\ell_{k,j} = x_i$ in C_j, then set $c(\ell_{k,j}) := c(x_{i,j})$, but if $\ell_{k,j} = \bar{x}_i$, then set $c(\ell_{k,j}) := c(\bar{x}_{i,j})$. If a clause gadget is of type A, then let $c(y_j) = c(y'_j)$ and $c(z_j) = c(z'_j)$ be two new colours. If a clause gadget is of type B, then let $c(z_j) = c(z'_j) = c(z''_j)$ be a new colour. If a clause gadget is of type C, then let $c(z_j) = c(z'_j)$ be a new colour. Lastly, all the vertices in T which do not belong to any gadget are given new colours. If T does not contain clause gadgets of type B, then its colour-multiplicity is 2, otherwise it is 3.

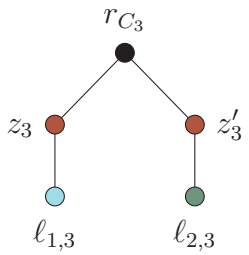
Note that Construction 4.1 can be done in polynomial time.



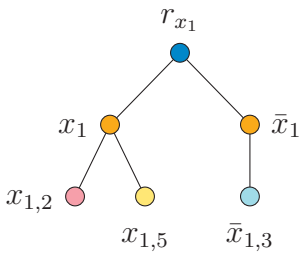
(a) Type A: Clause gadget of C_2



(b) Type B: Clause gadget of C_2



(c) Type C: Clause gadget of C_3



(d) Variable gadget of x_1

Figure 4.2: Examples of gadgets used in Construction 4.1.

Theorem 4.1

Colourful Components and Colourful Partition are NP-complete on coloured :

- quaternary 2-caterpillars with colour-multiplicity 3,
- ternary 3-caterpillars with colour-multiplicity 2, and
- binary 4-caterpillars with colour-multiplicity 2.

Proof. Obviously, the problem is in NP. Let ϕ be an instance of 3, 3-SAT with m clauses and n variables. We denote by m_3 the number of clauses containing 3 literals and by m_2 the number of clauses containing 2 literals. We transform ϕ into a coloured tree T as described in Construction 4.1. Note that T is either a quaternary 2-caterpillar, a ternary 3-caterpillar or a binary 4-caterpillar. The colour-multiplicity of T depends on the use of clause gadgets of type B, and therefore is exactly 3 for a quaternary 2-caterpillar but can be only 2 for a ternary 3-caterpillar or a binary 4-caterpillar. We claim that there exists a satisfying assignment of ϕ if and only if there is a set of exactly $n + 2m_3 + m_2$ edges in T whose removal makes T colourful. Let β be a satisfying assignment of ϕ. We define the set of edges S as follows:

- For each variable x_i, the set S contains the edge $\{r_{x_i}, x_i\}$ if $x_i = True$ in β, or $\{r_{x_i}, \bar{x}_i\}$ if $x_i = False$ in β .
- For each clause C_j with three literals:
  - If the clause gadget is of type A, then the set S contains two edges: one from the path between y_j and y'_j, and one from the path between z_j and z'_j. These edges are chosen in such a way that, in $G - S$, the leaf $\ell_{k,j}$ which belongs to the same connected component as the vertex r_{C_j} corresponds to (one of) the literal(s) satisfying the clause C_j in β.
  - If the clause gadget is of type B, then the set S contains two edges incident to the vertex r_{C_j} such that z_j, z'_j and z''_j are in different connected component in $G - S$. These edges are chosen so that, in $G-S$, the leaf $\ell_{k,j}$ which belongs to the same connected component as the vertex r_{C_j} corresponds to (one of) the literal(s) satisfying the clause C_j in β.
- For each clause C_j with two literals, and then represented by a clause gadget of type C, the set S contains either the edge $\{r_{C_j}, z_j\}$ or the edge $\{r_{C_j}, z'_j\}$. Again, this edge is chosen in order that, in $G - S$, the leaf $\ell_{k,j}$ which belongs to the same connected component as the vertex r_{C_j} corresponds to (one of) the literal(s) satisfying the clause C_j in β.

Clearly, the set S contains $n + 2m_3 + m_2$ edges. We denote by $\mathcal{F}$ the forest $T - S$, and by T' the connected component in $\mathcal{F}$ containing the root r. Obviously, two vertices of the same colour from a same variable gadget do not both belong to a same connected component of $\mathcal{F}$, and the same is true for a clause gadget. Also, note that two vertices of different variable gadgets do not have the same colour, and similarly for vertices of different clause gadgets. Lastly, observe that two vertices of two different gadgets belong to the same connected component if and only if they are connected through the backbone, which is in T'. Thus, if there exist two vertices of the same colour in a same connected component of $\mathcal{F}$, one is from a variable gadget and the other one from a clause gadget, and they both belong to T'. Without loss of generality, consider $x_{i,j}$ from the variable gadget of x_i and $\ell_{k,j}$ from the clause gadget of C_j, such that $x_{i,j}, \ell_{k,j} \in T'$. To prove a contradiction, assume that $c(x_{i,j}) = c(\ell_{k,j})$. Note that the literal represented by $\ell_{k,j}$ is $x_{i,j}$, otherwise the two vertices would not have the same colour. Since $\ell_{k,j}$ is in T', it is connected to the vertex r_{C_j} of the clause gadget, hence $\ell_{k,j}$ satisfies

the clause C_j. Therefore, the variable $x_i = True$ in β. By construction, this implies that the edge $\{r_{x_i}, x_i\}$ belongs to S, and that the subtree T_{x_i}, containing $x_{i,j}$, is not part of T', which is a contradiction.
Let S be a solution to Colourful Components on T. Observe that S contains at least one edge per variable gadget to put the vertices x_i and $\bar{x}_i$ into different connected components, at least two edges per clause gadget of type A or type B, and at least one edge per clause gadget of type C. Hence, $|S| \geq n + 2m_3 + m_2$. Suppose that $|S| = n + 2m_3 + m_2$, and therefore that S contains exactly one edge per variable gadgets, exactly two edges per clause gadget of type A or type B, and exactly one edge per clause gadget of type C. We denote by T' the connected component of $T - S$ containing the root r. Notice that, for each variable gadget, either x_i or $\bar{x}_i$ belongs to T', but not both. Also, for each clause gadget, exactly one leaf $\ell_{k,j}$ belongs to T'. We construct an assignment β of ϕ such that, for each variable gadget, if $\{r_{x_i}, x_i\} \in S$, then we set $x_i := True$ in β, and if $\{r_{x_i}, \bar{x}_i\} \in S$, then we set $x_i := False$ in β. To prove a contradiction, assume that there is a clause C_j which is not satisfied in ϕ with regard to β. Consider the leaf $\ell_{k,j} \in T'$ from the gadget clause of C_j, and assume without loss of generality that $\ell_{k,j} = x_i$. If C_j is not satisfied, then the variable $x_i := False$ in β. This means that S contains the edge $\{r_{x_i}, \bar{x}_i\}$, but not the edge $\{r_{x_i}, x_i\}$, and thus $x_{i,j} \in T'$. However, since $c(x_{i,j}) = c(\ell_{k,j})$, then S is not a solution for T, a contradiction. ■

4.2.2. The easy case

A *cyclic* 1-*caterpillar* is a connected graph with a unique cycle called the backbone such that each vertex either belongs to the backbone or is pendant. To simplify the notations, 1-caterpillars and cyclic 1-caterpillars are called *caterpillars*. In this subsection, we prove that Colourful Components and Colourful Partition can be solved in linear time on coloured caterpillars with unbounded maximum degree.

We consider the vertices in the backbone as internal vertices of stars, hence vertices of degree 1 are the leaves of a star whose internal vertex belongs to the backbone. We assume that the edges and the vertices in the backbone are either linearly or cyclically ordered, if the backbone is a path or a cycle, respectively.

Remark 4.1. Consider a coloured caterpillar. If two vertices of a star have the same colour, then at least one of these vertices is a leaf and it must belong to a different colourful component than the internal vertex of the star. Hence, a

caterpillar can be preprocessed in such a way that, for each such leaf, we add its adjacent edge to a set S_p. This procedure is repeated until there is no such leaf in $G - S_p$. At the end of the preprocessing, each star in $G - S_p$ is a *colourful star*, that is, only contains vertices with different colours. See an example of preprocessing in Fig. 4.3.

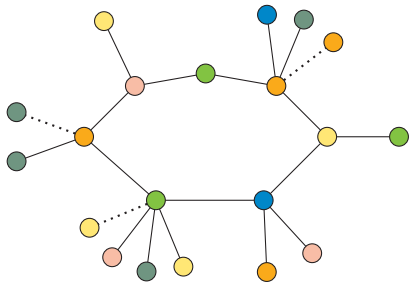


Figure 4.3: A 7-coloured cyclic caterpillar with only colourful stars. Dotted edges belong to all solutions to Colourful Components (up to isomorphism) and are removed from the graph in the preprocessing.

If a coloured caterpillar is not colourful, then it contains either one or at least two colours that appear more than once. We deal with these two cases independently in the following lemmas.

Lemma 4.1

Colourful Components and Colourful Partition can be solved in linear time on coloured caterpillars where exactly one colour appears at least twice.

Proof. Let $G = (V, E)$ be a coloured caterpillar such that there is exactly one colour γ appearing at least twice in G and $f^\gamma(G)$ denote the number of vertices of colour γ in G.

First, notice that if the backbone of G is a path, then an optimal solution to Colourful Partition with $f^\gamma(G)$ parts, *i.e.* with exactly one vertex of colour γ per part, can be found in linear time. Of course, the $f^\gamma(G)-1$ edges with endpoints in different parts form an optimal solution to Colourful Components.

Suppose now that the backbone is an induced cycle. Let $f_B^\gamma(G)$ be the number of vertices of colour γ that appear on the backbone. It is easy to see that if $f_B^\gamma(G) = 1$, then an optimal solution to Colourful Components contains exactly $f^\gamma(G) - 1$ edges. We show that if $f_B^\gamma(G) \geq 2$, then an optimal solution to Colourful Components contains exactly $f^\gamma(G)$ edges. The property is obviously true if $f^\gamma(G) = f_B^\gamma(G)$, that is, if all the vertices

with colour γ are on the backbone. Suppose that the property is true for $f^\gamma(G) = f_B^\gamma(G) + k$, for some positive integer k, and consider the case where $f^\gamma(G) = f_B^\gamma(G) + k + 1$. Let u be a vertex of colour γ that is not on the backbone, v be the neighbour of u and $G' = G - \{u\}$. For any optimal solution S' to Colourful Components on G', the connected component of the vertex v in $G' - S'$ contains a vertex of colour γ, otherwise S' is not optimal. Thus, any optimal solution on G contains at least one more edge than S'. Since u is a leaf, the set $S' \cup \{u, v\}$ is an optimal solution on G of size $f^\gamma(G') + 1 = f^\gamma(G)$. Of course, the partition of the vertices described by each connected component is an optimal solution to Colourful Partition since each part contains exactly one vertex of colour γ. ■

Lemma 4.2

Let G be a coloured caterpillar with only colourful stars such that at least two colours appear at least twice in G. Then there exists an optimal solution S of Colourful Components in G such that $S \subseteq B$, where B is the backbone of G.

Proof. We assume that $G = (V, E)$. Let $f(G) := |\{u \in V \mid \exists v \in V, v \neq u, c(v) = c(u)\}|$ denote the number of vertices whose colour appears at least twice in G. Notice that if at least two colours appear at least twice in G, then $f(G) \geq 4$. One can easily check that if $f(G) = 4$, then there always exists an optimal solution $S \subseteq B$ on G, so we suppose $f(G) \geq 5$.
Assume that for any coloured caterpillar with only colourful stars H with backbone B_H such that at least two colours appear at least twice, if $f(H) \leq t$, $t \geq 5$, then there exists an optimal solution $S_H \subseteq B_H$. Suppose that $f(G) = t + 1$, and take a vertex $u \in V$ such that there exists another vertex $v \in V$ of the same colour. Let G' denote the graph G in which the colour of u is set to a new colour, not yet used in the graph, hence $f(G') \leq t$. Let $S_{G'}$ be an optimal solution on G'. If $S_{G'}$ is also a solution on G, then obviously $S_{G'}$ is optimal on G, and the property that $S_{G'} \subseteq B_G$ holds. Therefore, we suppose there is no optimal solution on G' that is a solution on G. Notice that, in the connected component of u in $G - S_{G'}$, there is only one vertex v such that $c(v) = c(u)$. Let P be the bad path from u to v, and choose $e \in P \cap B$. Thus, $S_G := S_{G'} \cup \{e\}$ is an optimal solution of size $|S_{G'}| + 1$ on G and $S_G \subseteq B$. ■

Algorithm 2: From coloured caterpillar to ordered pairs.

Input: $G = (V, E)$, an ℓ-coloured caterpillar with only colourful stars and backbone B.

Output: A, a multiset of ordered pairs of vertices.

```
// Initialisation
```

let $U := \{u \in e \mid e \in B\}$ be an ordered set of vertices, w.r.t. the order on B;
let A be an empty multiset of ordered pairs of vertices;
let L be an array of length ℓ initialised at $NULL$;
let $proceed := True$;
let $u \in U$ such that, if U is linearly ordered, then u is minimum in U;

```
// if U is cyclically ordered, any u ∈ U can be taken
```

let $end := NULL$ and $temp\_end := NULL$;

```
// Main procedure
```

while *proceed* **do** `// O(n)`
  foreach $v \in \{w \in N[u] \mid d(w) = 1 \text{ or } w = u\}$ **do**
    if $L[c(v)] = NULL$ **then**
      $temp\_end := u$;
    else if $L[c(v)] \neq u$ **then**
      add $(L[c(v)], u)$ to A;
      if $u = end$ **then**
        $proceed := False$; `// U is cyclically ordered`
    $L[c(v)] := u$;
  $end := temp\_end$;
  if *U is linearly ordered and u is maximum in U* **then**
    $proceed := False$;
  $u := v$, such that v is the element after u in U, w.r.t. its order;
return A;

Let G be a coloured caterpillar with backbone B and only colourful stars. We say that a bad path P between two vertices of colour γ in G is a *colour-critical bad* path if and only if there is no other bad path P' between two vertices of colour γ such that $P' \cap B \subseteq P$. Hence, two colour-critical bad paths with endpoints of colour γ do not have any common edge in the backbone B.

Remark 4.2. Let G be a coloured caterpillar with only colourful stars such that at least two colours appear twice. We denote by B the backbone of G. Lemma 4.2 guarantees that there exists an optimal solution S to Colourful Components on G such that $S \subseteq B$. It is clear that if each colour-critical bad path contains an edge in S and $S \subseteq B$, then S is a solution to Colourful Components on G. Hence, there is an optimal solution to Colourful Components that contains only edges in B that also belong to some colour-critical bad path.

Now, the idea is to define a circular-arc graph H (an intersection graph of a collection of arcs on the circle) based on the colour-critical bad paths of G. A minimum clique cover $\mathcal{Q}$ of H, which is a partition of the vertex set into a minimum number of cliques, can be obtained in linear time [55]. We show that $\mathcal{Q}$ can be translated back into an optimal solution to Colourful Components and Colourful Partition on G in linear time.

Lemma 4.3

Let G be a coloured caterpillar with only colourful stars, and A be the multiset of pairs returned by Algorithm 2. Then there is a bijection between the set of colour-critical bad paths in G and the multiset A.

Proof. Let B be the backbone of $G = (V, E)$. A colour-critical bad path P from a to b is detected in Algorithm 2 at Line 11, when b is found to have the same colour γ as a (the last recorded vertex of colour γ). Let x be the internal vertex of the star to which a belongs, and y for b, respectively. When b is considered in the algorithm, the pair $(L[c(b)], y)$ is added to A at Line 12, and since $L[c(b)] = x$, then $(x, y) \in A$. Therefore, the arc with endpoints $(x, y) \in A$ corresponds to the colour-critical bad path P from a to b in G.

An ordered pair (x, y) in A refers to two vertices x and y in V that are internal vertices of two stars. If such a pair exists, then there are two vertices a and b with the same colour γ, such that a belongs to the same star as x and b to the same star as y, and in the path P from a to b, with regard to the order on B, there is no other vertex w with colour γ in a star whose internal vertex is in P (since the last seen vertex of colour γ, before b, is $L[c(b)] = a$). Thus, the path P is a colour-critical bad path and it corresponds to the pair (x, y) in A. ■

Lemma 4.4

Algorithm 2 runs in linear time.

Proof. Let G be a coloured caterpillar with only colourful stars and backbone B, and let A be the multiset of ordered pairs obtained by Algorithm 2 with input G.

In Algorithm 2, when a colour is detected for the first time at Line 9, the internal vertex u of the star is stored in the variable *end*. If the backbone induces a cycle, *i.e.* if G is a cyclic caterpillar, the second time that the vertex

end is considered in the main loop the algorithm sets the variable *proceed* to false at Line 14. If the backbone is a path, *i.e.* if G is a caterpillar, the algorithm considers each vertex exactly once and sets the variable *proceed* to false at Line 18. Thus, Algorithm 2 runs in linear time. ■

In the following, we show that we can reduce our problem on coloured caterpillars to the problem of finding a minimum clique cover in a circular-arc graph.

Definition 4.1: Clique cover

A *clique cover* of a graph $G = (V, E)$ is a partition $\mathcal{Q}$ of V such that for each $Q_i \in \mathcal{Q}$, Q_i is a clique. A *minimum clique cover* is a clique cover containing a minimum number of cliques.

Definition 4.2: Circular arc graph

A *circular-arc graph* is the intersection graph of a set of arcs on the circle, that is, each vertex of the graph can be represented as an arc on the circle and there is an edge between two vertices if and only if the corresponding two arcs intersect.

Theorem 4.2

Colourful Components and Colourful Partition can be solved in linear time on coloured caterpillars.

Proof. Let $G = (V, E)$ be a coloured caterpillar with backbone B. First, we prove that a solution to Colourful Components on G can be found in linear time. We apply the preprocessing to G, as defined in Remark 4.1, and denote by S_p the set of edges that have been removed. Hence, $G - S_p$ contains only colourful stars. If $G - S_p$ is colourful, then S_p is an optimal solution to Colourful Components. Otherwise, denote by $G' = (V', E')$ the connected component of $G - S_p$ which contains the backbone B. If G' contains exactly one colour that appears more than once, then according to Lemma 4.1 Colourful Components and Colourful Partition can be solved in linear time. Therefore, we assume that G' contains at least two colours that appear at least twice. Let A be the multiset of ordered pairs obtained by Algorithm 2 with input G'. According to Lemma 4.4, A can be obtained in linear time.

According to Lemma 4.3, each ordered pair (x, y) in A corresponds to a colour-critical bad path P from x to y in G. These paths can be seen as arcs on the circle, which represent a circular-arc graph $H = (X, F)$. Let $\mathcal{Q}$ be a minimum clique cover of H obtained in linear time with the use of the algorithm of Hsu and Tsai [55]. As mentioned in the paper, there are two types of cliques in a circular-arc graph. The first type of cliques contains three arcs which do not contain a common point of the circle. The second type of cliques contain a common point of the circle and are referred as *linear cliques*. It is proved that it suffices to use linear cliques in a minimum clique cover unless the graph is a complete graph and the unique maximal clique is not linear. Hence, either all cliques in $\mathcal{Q}$ are linear or the unique clique in $\mathcal{Q}$ is not linear. In the following, we first deal with the case where $\mathcal{Q}$ contains linear cliques and then consider the case where the unique clique in $\mathcal{Q}$ is not linear.

Suppose that all cliques in $\mathcal{Q}$ are linear. Let S' be an empty set of edges. Choose a clique $Q_i \in \mathcal{Q}$. From our construction of H, each vertex $z \in Q_i$ corresponds to a colour-critical bad path P_z in G. Let $D_i := \bigcap_{z \in Q_i} P_z$, and notice that, since Q_i is linear, $|D_i \cap B| > 0$. Then, choose an edge $e \in D_i \cap B$, and add e to S'. We claim that, once each clique in $\mathcal{Q}$ has been processed, that is, when $|S'| = |\mathcal{Q}|$, the set S' is an optimal solution to Colourful Components on G. Notice that S' can be computed in linear time. As stated before, each colour-critical bad path in G' maps to an ordered pair in A, which corresponds to a vertex of H. Hence, a linear clique Q_i in H corresponds to a set of colour-critical bad paths sharing a common subpath D_i. The set S' contains an edge in $D_i \cap B$ for each $Q_i \in \mathcal{Q}$, hence there is no colour-critical bad path in $G' - S'$. Since $S' \subset B$ and each colour-critical bad path has an edge in S', then, following Remark 4.2, S' is a solution to Colourful Components on G'. Moreover, since $\mathcal{Q}$ is minimum, S' is an optimal solution on G'. Thus, the set $S := S_p \cup S'$ is an optimal solution to Colourful Components on G.

Now, suppose that the unique clique in $\mathcal{Q}$ is not linear, which implies that H is a complete graph. As mentioned before, there exist at least three arcs that do not have a common point on the circle. Let z denote one of these arcs such that it does not strictly contain any other arc. Clearly, since H is a complete graph, z overlaps every other arc. Also, since z does not strictly contain any other arc, if one extremity of z does not overlap another arc z', then its other extremity must overlap z'. Let P_z be the colour-minimal bad path, with endpoints u and v, corresponding to z. Denote

by u' and v' the neighbours of u and v in P_z, respectively. Then, the set $S' := \{\{u, u'\}, \{v, v'\}\}$ is a solution to Colourful Components on G'. Since the intersection of all the colour-critical bad paths in G' is empty, any solution to Colourful Components on G' contains at least 2 edges. Hence, S' is an optimal solution and $S := S_p \cup S'$ is an optimal solution to Colourful Components on G.

Finally, as $|E| \in O(|V|)$, we can detect each connected component of $G - S$ in linear time (for instance, with a breadth-first search). Thus, we can construct the partition π of V such that each part corresponds to a connected component of $G - S$ in linear time. Obviously, π is a solution to Colourful Partition on G. Since S is optimal, due to the structure of G, the partition π is optimal. ■

Note that if there exists a colour-critical bad path P such that $P \cap B \subset P'$ for some other colour-critical bad path P' in G, then P does not have to be represented in the circular-arc graph. While this observation can help decreasing the number of vertices in the circular-arc graph, it does not affect the overall worst-case linear complexity of the algorithm.

Figure 4.4 gives an example of a cyclic 1-caterpillar G and its representation as circular-arc graph H where a minimum clique cover on H represents an optimal solution to Colourful Components on G.

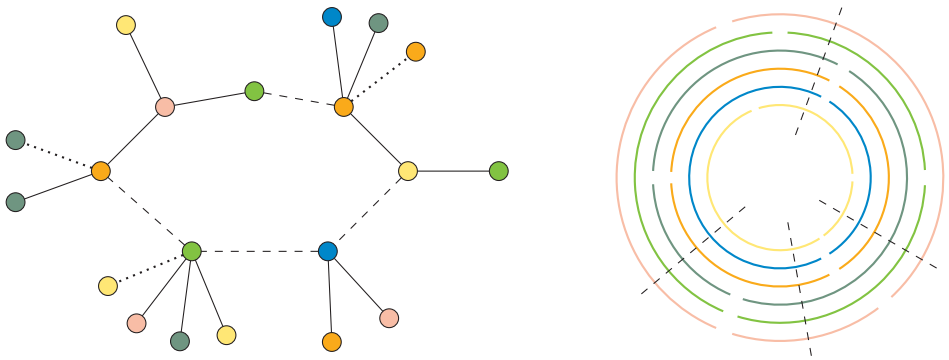


Figure 4.4: On the left, a cyclic 1-caterpillar G: dotted edges are removed in the preprocessing; dashed edges are obtain from the algorithm; dotted and dashed edges form an optimal solution to Colourful Components. On the right, an arc representation of the circular-arc graph constructed from the colour-critical bad paths in G (after preprocessing): dotted segments represent a minimum clique cover and correspond to the dashed edges in G.

4.3. Colourful Components on planar graphs

In [17], the authors prove that Colourful Components is NP-complete even when restricted to 3-coloured graphs with maximum degree 6. Using a similar reduction from Planar 3-SAT, we show how the vertices of degree 6 can be replaced with gadgets only containing vertices of degree 4, or 3, if we relax the number of colours from 3 to 5, or to 12, respectively.

Planar 3-SAT

Input: A 3-CNF formula ϕ in which the bipartite graph of variables and clauses is planar.

Question: Is there a satisfying assignment of ϕ?

Note that Planar 3-SAT is NP-complete [64] and it can be shown that it remains so even if each clause contains exactly 3 literals [83]. In the following, we consider the latter version.

Construction 4.2

Given an instance ϕ of Planar 3-SAT, that is a set of m clauses $C_1, C_2, \ldots, C_m$ on n variables, we construct the graph $G = (V, E)$ such that:

- For each variable x in ϕ, let m_x denotes the number of clauses in which x appears. We construct a cycle of length $4m_x$ in G with vertices $V_x := \{x_j^1, x_j^2, x_j^3, x_j^4 \mid x \in C_j\}$ with an arbitrary fixed cyclic ordering of the clauses containing x. The vertices are coloured alternatively with two colours c_o and c_e, that is, $c(x_j^1) = c(x_j^3) = c_o$ and $c(x_j^2) = c(x_j^4) = c_e$, for all j such that $x \in C_j$.
- For each clause C_j containing three variables p, q and r, we construct a clause gadget. We propose two types of gadgets:
  - The gadget $\mathcal{A}_j^4$ is made of a cycle of length 3, with vertices a_j^1, a_j^2 and a_j^3 such that each a_j^i is given colour i, different from c_o and c_e. We define how the vertices from V_p are connected to $\mathcal{A}_j^4$. If the variable p appears as a positive literal in C_j, connect the vertices p_j^1 to a_j^1 and p_j^2 to a_j^2. Otherwise, if p occurs as a negative literal, connect the vertices p_j^2 to a_j^1 and p_j^3 to a_j^2. Do the same for the

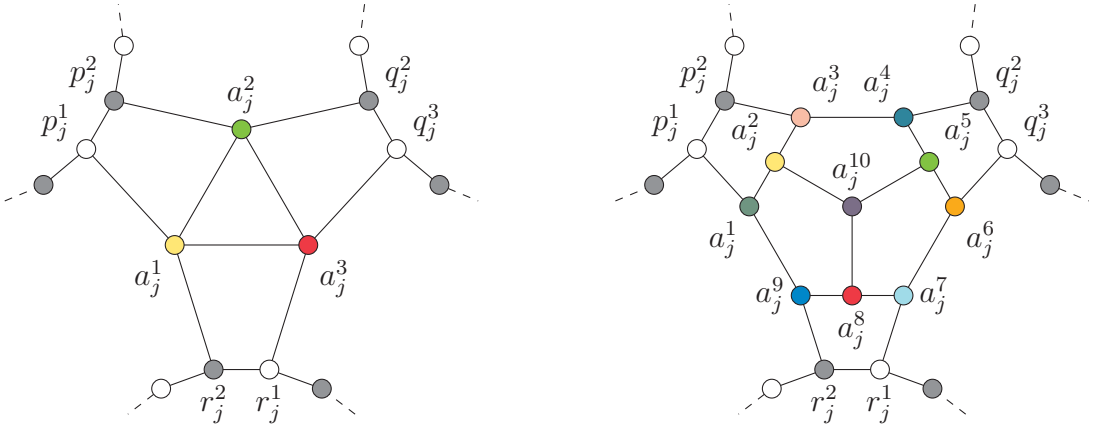
variables q and r by connecting the corresponding vertices in V_q to a_j^2 and a_j^3, and the corresponding vertices in V_r to a_j^3 and a_j^1. Notice that the vertices in $\mathcal{A}_j^4$ have degree 4.

– The gadget $\mathcal{A}_j^3$ is made of a cycle of length 9 with vertices labelled $a_j^1, \ldots, a_j^9$ and an additional vertex a_j^{10} connected to a_j^2, a_j^5 and a_j^8. We set the colour i to each vertex a_j^i, different from c_o and c_e. We define how the vertices from V_p are connected to $\mathcal{A}_j^3$. If the variable p appears as a positive literal in C_j, connect the vertices p_j^1 to a_j^1 and p_j^2 to a_j^3. Otherwise, the variable p occurs as a negative literal, connect the vertices p_j^2 to a_j^1 and p_j^3 to a_j^3. Do the same for the variables q and r by connecting the corresponding vertices in V_q to a_j^4 and a_j^6, and the corresponding vertices in V_r a_j^7 and a_j^9. Notice that the vertices in $\mathcal{A}_j^3$ have degree 3.

See Fig. 4.5 for an example of the gadgets.

Since the bipartite graph of variables and clauses of ϕ is planar and each vertex can be replaced by a clause or vertex gadget, with a correct cyclic ordering of the clauses for each variable, the resulting graph G is planar.

Note that Construction 4.2 can be done in polynomial time.



Gadget with vertices of degree 4. Gadget with vertices of degree 3.

Figure 4.5: Two clause gadgets $\mathcal{A}_j^4$ (left) and $\mathcal{A}_j^4$ (right) of a clause $C_j := (p \vee \bar{q} \vee r)$. White vertices have colour c_o, grey vertices have colour c_e.

Theorem 4.3

COLOURFUL COMPONENTS is NP-complete on 5-coloured planar graphs with maximum degree 4 and on 12-coloured planar graphs with maximum degree 3.

Proof. Let ϕ be an instance of PLANAR 3-SAT with m clauses and n variables and $G = (V, E)$ be the graph obtained through Construction 4.2, using either $\mathcal{A}_j^4$ gadgets or $\mathcal{A}_j^3$ gadgets. Notice that, depending on the type of gadget used, G is either a 5-coloured graph with maximum degree 4, or a 12-coloured graph with maximum degree 3. We claim that there exists a satisfying assignment β for ϕ if and only if there exists a solution S to COLOURFUL COMPONENTS such that $|S| = 6m + 4m = 10m$.

Let β be a satisfying assignment for ϕ. If a variable $x = True$ in β, then remove the edges $\{x_k^4, x_j^1\}$ and $\{x_j^2, x_j^3\}$, for each clause C_j, where C_k precedes C_j in the order of the clauses containing x. Otherwise, if $x = False$ in β, then remove the edges $\{x_j^1, x_j^2\}$ and $\{x_j^3, x_j^4\}$ for each clause C_j containing x. Hence, $\sum_{1 \leq i \leq n} 4m_i/2 = 6m$ edges have been removed, and G does not have any bad path that contains only vertices from the variable cycles. Now, choose a clause C_j and denote its variables by p, q and r. Without loss of generality assume that p satisfies C_j. Remove the 4 edges between the clause gadget of C_j and the vertices in V_q and V_r. Without loss of generality, assume that C_j contains p as a positive literal, hence the gadget is connected to the vertices p_j^1 and p_j^2. Since p is set to true in β, the edges $\{x_k^4, x_j^1\}$ and $\{x_j^2, x_j^3\}$ have been removed. Therefore, the vertices p_j^1, p_j^2 and the vertices in the clause gadget form a colourful component. The other case where C_j contains p as a negative literal is similar, and the vertices p_j^2, p_j^3 and those in the clause gadget form a colourful component. If the clause gadget is $\mathcal{A}_j^4$, then the colourful component is of size 5. If the clause gadget is $\mathcal{A}_j^3$, then the colourful component is of size 12. Since 4 edges are removed for each clause, a total of $4m$ edges between variable cycles and clause gadget are removed. Let S be the set of removed edges, and notice that S is a solution to COLOURFUL COMPONENTS such that $|S| = 6m + 4m = 10m$.

Let S be a solution to COLOURFUL COMPONENTS on G such that $|S| = 10m$. Since the vertices of the variable cycles are coloured alternatively with c_o and c_e, the set S must contain half of the edges of each variable cycle, thus S contains at least $\sum_{1 \leq i \leq n} 4m_i/2 = 6m$ edges from the variable cycles. Independently on which type of clause gadget has been used in Construction 4.2, we show how many edges S must contain for each clause

gadget. Choose a clause C_j and consider its clause gadget. It is clear that if S contains at most 2 edges from the clause gadget or between the clause gadget and its variable cycles, then $G - S$ is not colourful. If S contains 3 such edges, then at least 2 additional edges from the variable cycles must belong to S, otherwise there would exist a connected component in $G - S$ that contains vertices of at least two different variable cycles. However, it is impossible that S contains 5 edges per clause gadget or $|S| > 10m$. Finally, if S contains exactly 4 such edges, then necessarily S contains the edges between the clause gadget and two of the variable cycles, otherwise $G - S$ would not be colourful. Thus, since $|S| = 10m$, the set S contains exactly $4m$ edges between variable cycles and clause gadgets, and all edges in S are either from variable cycles or between variable cycles and clause gadgets. Let β be an assignment of ϕ described as follows. We consider that the edges in S have been removed from G. For each variable x and each clause C_j containing the literal x, if the edge $\{x_j^1, x_j^2\}$ belongs to S, then set $x := False$ in β. Otherwise, if the edge $\{x_j^2, x_j^3\}$ belongs to S, and therefore $\{x_j^1, x_j^2\}$ does not, then set $x := True$ in β. To prove that β is a satisfying assignment of ϕ, note that if a variable cycle of a variable x is connected to the clause gadget of a clause C_j, then C_j is satisfied by x. Suppose the opposite, and assume, without loss of generality, that x appears as a positive literal in ϕ but is set to $False$ in β. This means that the edge $\{x_j^1, x_j^2\}$ belongs to S, hence has been removed from G. Therefore, the edge $\{x_j^2, x_j^3\}$ belongs to G, since we suppose that the edges in the variable cycles are removed alternately. Then, x_j^1 and x_j^3 belong to the same colourful component, but $c(x_j^1) = c(x_j^3)$, a contradiction. We conclude that β is a satisfying assignment of ϕ. ■

4.4. Conclusion

The NP-completeness of the problems on 2-caterpillars with unbounded degree demonstrates the inherent complexity of the problems. We prove that both problems remain NP-complete on quaternary 2-caterpillars, ternary 3-caterpillars and binary 4-caterpillars, where both the maximum degree and the hair length are bounded by small constants. Nevertheless, our linear-time algorithm for both problems on general 1-caterpillars, with unbounded degree, generalises the class of paths and cycles, and beats the complexity of the previous best known algorithm

for paths. In order to complete the dichotomy on k-caterpillars, three questions remain: What is the complexity of the problems on binary 3-caterpillars, binary 2-caterpillars and ternary 2-caterpillars.

We also prove that Colourful Components is NP-complete on 5-coloured planar graphs with maximum degree 4 and on 12-coloured planar graphs with maximum degree 3. A natural question is to ask whether the problem remains NP-complete when the number of colours is decreased but the maximum degree is 3 or 4.

Chapter 5

Scheduling of Dial-A-Ride Problems with Soft Time Constraints

Outline

The Dial-a-Ride problem may contain various constraints for pickup-delivery requests, such as time windows and ride time constraints. For a tour, given as a sequence of pickup and delivery stops, there exist polynomial time algorithms to find a schedule respecting these constraints (provided that there exists one). However, if no feasible schedule exists, a natural question is to find a schedule minimising constraint violations. We model a generic fixed-sequence scheduling problem, allowing lateness and ride time violations with linear penalty functions and prove its APX-hardness. Then, we propose several polynomial-time algorithms for restricted types of instances.

Some of the results presented in this chapter appear in the following paper:

- J. Chlebíková, C. Dallard and N. Paulsen, 'Complexity of scheduling for DARP with soft ride times', in *International Conference on Algorithms and Complexity*, Springer, 2019, pp. 149–160. DOI: 10.1007/978-3-030-17402-6_13.

A journal version containing our latest results is under construction.

5.1. Introduction

The Dial-A-Ride Problem (DARP) is a well studied variant of the Vehicle Routing Problem. The general idea of the DARP is the design of routes and schedules for a fleet of vehicles serving customers with pickup and delivery requests. The DARP, with its various restrictions, serves as a model for many real-world problems in logistics, for instance, passenger transportation and pickup-delivery of perishable goods. For detailed reviews of DARPs, we refer the readers to [30, 54].

The study of the Dial-A-Ride Problem can be split into three main subproblems: the *clustering* of the requests into tours, the *routing* of the stops within each tour into a sequence, and the *scheduling* of the stops inside the tours [34]. These subproblems are the source of major research topics in operation research, each of them intensively studied. To get a better understanding of the inherent complexity of the problems, and eventually obtain faster algorithms, many restricted models have been studied.

In this chapter, we focus on the scheduling subproblem, where the input of the problem is a tour with fixed sequence of stops, a set of pickup-delivery requests, and time constraints with their corresponding penalty functions. Each pickup-delivery request is represented by two stops, the first one as a pickup and the second one as a delivery. The visit of each stop has to be performed within a given *time window*. Furthermore, the time between the scheduled pickup and delivery of a same request is bounded by a given *maximum ride time*.

Time window and ride time constraints are naturally arising when scheduling pickups and deliveries. When both constraints must be respected in the solution, there exist efficient algorithms [44, 81]. However, in all these approaches, ride time and time window constraints are *hard* in the sense that a solution must respect all the constraints.

In case there is no feasible schedule, one may look for a schedule "close" to a feasible one with minimal penalties. Therefore, variants of the problem with *soft* constraints, in which the violation of constraints is allowed but penalised, have been introduced. Depending on the type of constraints and their corresponding penalty functions, various results can be obtained. For instance, when the only constraints are time windows for the stops, Dumas, Soumis and Desrosiers [39] proposed a linear programming approach for convex penalty functions with a linear time complexity, but their algorithm does not incorporate ride time constraints. We refer the reader to [85] for a recent survey on scheduling (timing) problems given a fixed sequence of stops (tasks). To the best of our knowledge, the complexity of the problem with soft ride time constraints was previously unknown.

We propose a systematic study of the complexity of the problem when allowing time window and ride time constraints violations with linear non-decreasing penalty functions.

5.2. Problem statement

For $\triangle \in \{\leq, <, \geq, >\}$, and X a well ordered set, let $X_{\triangle x} := \{y \in X : y \triangle x\}$ and $X[i]$ be the i-th smallest element of X. We consider that all times are natural numbers.

We are given a sequence $\mathsf{S} = (1, 2, \ldots, 2n)$, $n \in \mathbb{N}$, of $2n$ stops in the order in which their visits must be scheduled (in case of no ambiguity $s \in \mathsf{S}$ also represents an positive integer).

Each stop $s \in \mathsf{S}$ is associated with a pair of times (a_s, b_s), $0 \leq a_s \leq b_s$, representing a time window in which a visit of the stop s should take place (we talk about time window constraint). Without loss of generality we suppose that $a_1 = 0$.

Furthermore, we have a set P of n requests representing the pairs (p, d) of stops from S, $p < d$, where p is a pickup and d is a delivery stop. Each request (p, d) has a time constraint $r_{p,d}$ ($r_{p,d} \geq 0$) on the maximum ride time (we talk about ride time constraint): a visit at stop d should be scheduled at most $r_{p,d}$ time units after the visit at stop p. Each stop s serves exactly one request (either as a pickup or as a delivery stop) and all times are represented as non-negative integers.

As it has been mentioned in Section 5.1, it is not always possible to schedule the visits for all stops (in a given order) with respect to their time windows and ride time constraints. Therefore we introduce the model in which the lateness and ride time constraints can be violated at a cost (soft constraints). In our model, the penalty functions are linear and non-decreasing.

In order to model soft constraints, each stop $s \in \mathsf{S}$ is associated with a penalty function $\sigma_s^L : \mathbb{N} \to \mathbb{Q}$, mapping visit times which are later than the time window bounds to a non-negative penalty. The function $\sigma_s^L(x)$ is such that $\sigma_s^L(x) = 0$ for $x \leq b_s$ and otherwise $\sigma_s^L(x) = \alpha_s^L \cdot (x - b_s) + \beta_s^L$, for given $\alpha_s^L, \beta_s^L \in \mathbb{Q}_{\geq 0}$. Also, each request $(p, d) \in \mathsf{P}$ is associated with a penalty function $\sigma_{p,d}^{RT} : \mathbb{N} \to \mathbb{Q}$, mapping ride times exceeding the given maximum ride times to a non-negative penalty. The function $\sigma_{p,d}^{RT}$ is such that $\sigma_{p,d}^{RT}(x) = 0$ for $x \leq r_{p,d}$ and otherwise $\sigma_{p,d}^{RT}(x) = \alpha_{p,d} \cdot (x - r_{p,d}) + \beta_{p,d}$, for given $\alpha_{p,d}, \beta_{p,d} \in \mathbb{Q}_{\geq 0}$.

Note that there is no penalty function for scheduling a stop earlier than the opening time of its time window, as our model does not allow this case. However, we show in Lemma 5.1 that earliness at stops can be expressed with a linear number of ride time constraints. Hence, we can assume that for each stop $s \in \mathsf{S}$, if s is visited at time x, then $x \geq a_s$. When we say "soft time window" we imply that the stop can be scheduled later than the time window, but not earlier.

A *schedule* $\mathsf{t} = (\mathsf{t}_1, \ldots, \mathsf{t}_{2n})$ is a sequence of visit times for all the stops in S (in the given order), where we say that t schedules a stop $s \in \mathsf{S}$ at time t_s. We say that a schedule t is *feasible* if and only if for all stops $s \in \mathsf{S}$, $\mathsf{t}_s \geq a_s$ and for any $s \in \mathsf{S}_{<2n}$, $\mathsf{t}_s \leq \mathsf{t}_{s+1}$. Obviously, each time window and ride time constraint is either *violated* or *satisfied* by a schedule t. The *cost* $c(\mathsf{t})$ of a schedule t is the sum of the penalties of violated constraints:

$$c(\mathsf{t}) = \sum_{s \in \mathsf{S}} \sigma_s^L(\mathsf{t}_s) + \sum_{(p,d) \in \mathsf{P}} \sigma_{p,d}^{RT}(\mathsf{t}_d - \mathsf{t}_p)\,. \tag{5.1}$$

When we solve the Min Pickup-Delivery Scheduling problem, we look for a feasible schedule t with a minimum cost.

Min Pickup-Delivery Scheduling (Min PDS)

Input: An instance I of Min PDS.
Output: A feasible schedule t of I such that $c(\mathsf{t})$ is minimum.

Overview of the results.
We investigate how penalisation of the time window and ride time constraints contributes to the computational complexity of the problem. We show that an essential factor for the complexity are soft ride time constraints. In Section 5.2.1, we give some remarks on our model. An overview of complexity results is shown in Table 5.1. We prove the NP-hardness of Min Pickup-Delivery Scheduling even with hard time window constraints in Section 5.3.1. On the other hand, we show that the problem can be solved in polynomial time in case of hard ride time constraints in Section 5.3.2. Further underlining the role of ride time constraints, in Section 5.4, we give a parameterized algorithm that solves Min Pickup-Delivery Scheduling in polynomial time if all maximum ride time are bounded by a constant. Finally, in Section 5.5, we show that some structural properties in the sequence of the stops can be exploited to find a polynomial-time algorithm. Namely, we present an $O(n^4)$ time algorithm when all pickups precede all the deliveries in the sequence.

Ride time constraints	*Time window constraints*	
	Hard	Soft
Hard	$O(n)$ [44]	$O(n)$ [Theorem 5.3]
0, $\beta_{p,d} = 0$	$O(n)$	$\leftarrow$ $O(n)$ [Proposition 5.1]
Soft, bounded values	P	$\leftarrow$ P [Theorem 5.4]
Soft, unbounded values	NP-hard, APX-hard Theorems 5.1 and 5.2	$\rightarrow$ NP-hard, APX-hard

Table 5.1: Overview of complexity results classified by constraints. Arrows mean results are inferred.

5.2.1. Remarks on the model

Express earliness through ride time constraints.
As we mentioned earlier, our model does not allow earliness at stops. However, we show that this is not restrictive.

Lemma 5.1

An instance I with $2n$ stops where earliness is allowed can be modified into an instance I' with $6n$ stops where earliness is not allowed such that, for any feasible schedule t of I, there exists a feasible schedule t' of I' with $c(t') = c(t)$.

Proof. We denote by σ_s^E the linear non-decreasing penalty function associated with the earliness at stop s.
First, let $B := \max_{s \in \mathsf{S}} b_s$. Choose a stop $s \in \mathsf{S}$. We create two new stops s_p and s_d, insert s_p in S just before s and append s_d at the end of S. We set $a_{s_p} := 0$, $b_{s_p} := b_s$, $a_{s_d} := B$ and $b_{s_d} := B$. The penalty function associated with the lateness at stop s_p is $\sigma_{s_p}^L(x) = 0$ (since the stop s is already penalised for lateness). The function $\sigma_{s_d}^L(x)$ can also be set to 0. Then, we create the request (s_p, s_d), which we add to P, and we set $r_{s_p,s_d} := B - a_s$. We set the penalty function of the ride time constraint as follows: $\sigma_{s_p,s_d}^{RT}(x) := 0$ if $x \leq r_{s_p,s_d}$ and otherwise $\sigma_{s_p,s_d}^{RT}(x) := \sigma_s^E(B - x)$. Lastly, we modify the time window of the stop s and set $a_s := 0$. Obviously, the stops s_d can always be scheduled at time B, thus the earliness at stop s is now penalised by the penalty function of the ride time constraint of the request (s_p, s_d). When applied to all $2n$ original stops, the resulting instance contains $6n$ stops. ■

Driving times, (un)loading times.
In favour of simplicity, our model neglects times needed to travel between stops as well as loading or unloading times. We emphasise that this is not restrictive, since we focus on the scheduling of fixed sequences. In the following, we show that incorporating these times into an extended problem definition is not more expressive. Suppose X is an instance of the *extended* Min PDS, where for each stop $s \in \mathsf{S}$ we are additionally given a driving time Γ_s to reach stop $s+1$ (assume $d_{2n} = 0$ for the last stop) and (un)loading time v_s at stop s. Further, a schedule for X is only feasible if for $s \in \mathsf{S}_{<2n}$ it holds the strengthened inequality $\mathsf{t}_s + v_s + d_s \leq \mathsf{t}_{s+1}$. Lastly, lateness penalties and ride time penalties factor in the visiting times, such that the cost $\widehat{c}(\mathsf{t})$ of a schedule t of X is defined as

$$\widehat{c}(t) := \sum_{s \in \mathsf{S}} \sigma_s^L(\mathsf{t}_s + v_s) + \sum_{(p,d) \in \mathsf{P}} \sigma_{p,d}^{RT}(\mathsf{t}_d + v_d - \mathsf{t}_p)\,.$$

Lemma 5.2

Let X be an instance of the *extended* Min PDS and I_X its corresponding instance of Min PDS. Then there exists an one-to-one cost-preserving mapping between the schedules of X and I_X. Furthermore, I_X can be constructed in linear time.

Proof. For a stop $\ell \in \mathsf{S}$ denote $\Gamma_\ell := \sum_{s<\ell}(d_s + v_s)$. Define an instance I_X of Min PDS with the same sequence S of the stops, the same set of requests, and the same penalty function coefficients as X. For all $s \in \mathsf{S}$ set time windows as $a'_s := a_s - \Gamma_s$ and $b'_s := b_s - \Gamma_s - v_s$. The maximal ride times are defined as $r'_{p,d} := r_{p,d} - \Gamma_d + \Gamma_p - v_d$ for all $(p,d) \in \mathsf{P}$. Note that all adjustments of this kind can be performed in a single preprocessing pass over the extended instance X in linear time. We prove that there is a one-to-one correspondence between schedules of X and I_X. For t a schedule of X, let $f(\mathsf{t})$ map to a schedule $\mathsf{t}' = f(\mathsf{t})$ of I_X such that for $s \in \mathsf{S}$, $\mathsf{t}'_s = \mathsf{t}_s - \Gamma_s$. Note that f is bijective.
First, we show that t is feasible for X exactly when t' is feasible for I_X. For $s \in \mathsf{S}$ we have $\mathsf{t}_s \geq a_s \Leftrightarrow \mathsf{t}_s - \Gamma_s \geq a_s - \Gamma_s \Leftrightarrow \mathsf{t}'_s \geq a'_s$; for $s \in \mathsf{S}_{<2n}$ we have

$$\begin{aligned} & \mathsf{t}_s + v_s + d_s \leq \mathsf{t}_{s+1} \\ \iff\ & \mathsf{t}_s - \Gamma_s = \mathsf{t}_s + v_s + d_s - \Gamma_{s+1} \leq \mathsf{t}_{s+1} - \Gamma_{s+1} \\ \iff\ & \mathsf{t}'_s \leq \mathsf{t}'_{s+1}\,. \end{aligned}$$

Now, we show that the penalty cost of t for X equals the penalty cost of t' for I_X. We have

$$\begin{aligned}
\widehat{c}(\mathsf{t}) &= \sum_{s\in\mathsf{S}} \sigma_s^L(\mathsf{t}_s + v_s) + \sum_{(p,d)\in\mathsf{P}} \sigma_{p,d}^{RT}(\mathsf{t}_d + v_d - \mathsf{t}_p) \\
&= \sum_{\substack{s\in\mathsf{S} \\ \mathsf{t}_s+v_s>b_s}} \alpha_s(\mathsf{t}_s + v_s - b_s) + \beta_s + \sum_{\substack{(p,d)\in\mathsf{P} \\ \mathsf{t}_d+v_d-\mathsf{t}_p>r_{p,d}}} \alpha_{p,d}(\mathsf{t}_d + v_d - \mathsf{t}_p - r_{p,d}) + \beta_{p,d} \\
&= \sum_{\substack{s\in\mathsf{S} \\ \mathsf{t}'_s>b'_s}} \alpha_s(\mathsf{t}'_s - b'_s) + \beta_s + \sum_{\substack{(p,d)\in\mathsf{P} \\ \mathsf{t}'_d-\mathsf{t}'_p>r'_{p,d}}} \alpha_{p,d}(\mathsf{t}'_d - \mathsf{t}'_p - r'_{p,d}) + \beta_{p,d} \qquad (5.2) \\
&= \sum_{s\in\mathsf{S}} \sigma_s^L(\mathsf{t}'_s) + \sum_{(p,d)\in\mathsf{P}} \sigma_{p,d}^{RT}(\mathsf{t}'_d - \mathsf{t}'_p) = c(\mathsf{t})
\end{aligned}$$

To verify Eq. (5.2), note that $\mathsf{t}_s + v_s - b_s = \mathsf{t}'_s - b'_s$ and $\mathsf{t}_d + v_d - \mathsf{t}_p - r_{p,d} = \mathsf{t}'_d - \mathsf{t}'_p - r'_{p,d}$ hold. These can be applied to the penalty function arguments as well as the sum ranges. ■

Waiting times.

Constraints w_s on the waiting times, *i.e.* maximum times to wait between two consecutive stops s and $s+1$ (as in [44]), are omitted in our model since they can be expressed by ride time constraints. Assume that, for a stop $s \in \mathsf{S}_{<2n}$, the constraint $\mathsf{t}_s + w_s \geq \mathsf{t}_{s+1}$ is given for any schedule t, $w_s \geq 0$. Simply insert two additional stops: the stop p immediately before s and the stop d immediately after $s+1$ into S and add a request (p,d) to P with $r_{p,d} = w_s$. Replacing all waiting time constraints leads to an equivalent instance with at most $6n-2 \in O(n)$ requests.

Increasing time windows opening times.

As earliness is not allowed in our model, we expect that any instance of $2n$ stops has $a_s \leq a_{s+1}$ for all $s \in \mathsf{S}_{<2n}$. If for a stop s, $s \in \mathsf{S}_{<2n}$, we have $a_s > a_{s+1}$, for any feasible schedule t it holds $\mathsf{t}_s \geq a_s$ and $\mathsf{t}_{s+1} \geq \mathsf{t}_s$ and therefore $\mathsf{t}_{s+1} < a_s$ cannot hold for any feasible scheduling. We can therefore preprocess the instance in such a way that for all $s \in \mathsf{S}_{<2n}$, $a_{s+1} := \max\{a_s, a_{s+1}\}$. Notice that due to this property and the fact that the last stop $2n$ is a delivery stop, it always exists an optimal schedule t^* such that $\mathsf{t}^*_{2n} = a_{2n}$.

As soon as possible deliveries.
All deliveries can be scheduled at a time as soon as possible without increasing costs. Let d be a delivery of a request $(p, d) \in \mathsf{P}$ and t a feasible schedule. We define a schedule t' with $\mathsf{t}'_s := \mathsf{t}_s$ for all stops $s \in \mathsf{S} \setminus \{d\}$ and $\mathsf{t}'_d := \max\{a_d, t_{d-1}\}$. Clearly, t' is feasible. Obviously, t'_d can only decrease the lateness at d as well as the ride time for $(p, d) \in \mathsf{P}$ with no changes in scheduling of the other stops.

5.3. Complexity study

5.3.1. Scheduling with soft ride time constraints

First, we study the variant of Min Pickup-Delivery Scheduling in which ride time constraints may be violated in return for a penalty (soft constraints), but the time windows must be respected (hard constraints). We show that the problem is NP-hard and APX-hard even in case of restricted time windows and very circumscribed penalty functions. The proof is based on a reduction from the Maximum Dicut problem which is known to be NP-hard and APX-hard when restricted to directed acyclic graphs (DAG) [48, 63].

First, we show that hard time window constraints can be expressed as soft time window constraints with "big penalty".

Remark 5.1. The idea is to set the penalties for lateness at each stop to such values that any optimal schedule must respect the time windows. Let I be an instance of Min PDS with hard time windows. As mentioned in Section 5.2.1, there exists an optimal schedule t of I such that $\mathsf{t}_{2n} = a_{2n}$. Therefore, the actual maximum ride time of each request is bounded by the value a_{2n}. Let $M = \max_{(p,d)\in\mathsf{P}} \sigma^{RT}_{p,d}(a_{2n})$. Then the instance I can be transformed into an instance I' of Min PDS with soft time windows by setting $\alpha_s = 0$ and $\beta_s = nM + 1$ for all $s \in \mathsf{S}$ (hence $\sigma^L_s(x) = nM + 1$ for $x > b_s$). The cost of any schedule respecting time window constraints is at most nM, and hence there always exists a schedule of I' with cost strictly less than $nM + 1$.

A directed cut (A, B) of a directed graph $G = (V, E)$ is a partition of V into two subsets A, B. Its size $s(A, B) := |\{(u, v) \in E \mid u \in A, v \in B\}|$ is the number of outgoing arcs from A to B. The Maximum Dicut problem is defined as follows:

Maximum Dicut

Input: A directed graph $G = (V, E)$.
Task: Find a directed cut (A, B) of maximum size in G.

Construction 5.1

Let $G = (V, E)$ be a connected directed acylic graph with $|V| = n$ and $|E| = m$. Since G is a DAG, the vertices of G can be labelled by $1, 2, \ldots, n$ in a topological ordering in such a way that for any arc $(u, v) \in E$ it holds $lab(u) < lab(v)$, where $lab(z)$ represents the number used for labelling the vertex z [82].

We show how the graph G can be transformed into an instance I of Min PDS. The sequence of stops for I is defined as the concatenation $\mathsf{S} := \mathsf{S}^1\mathsf{S}^2 \ldots \mathsf{S}^n$, where each S^v represents a gadget of stops for each vertex $v \in V$. Let $v \in V$ be fixed, then the gadget S^v contains the stop s_e^v for each arc e of G incident to v, that is, $e = (v, u)$ or $e = (u, v)$ for $u \in V$. The stops within the gadget S^v are ordered in such a way that all stops belonging to outgoing arcs precede all stops belonging to ingoing arcs. Note that S has $2m$ stops. For each vertex $v \in V$ the time windows of all stops $s \in \mathsf{S}^v$ are set to $a_s := lab(v) - 1$ and $b_s := lab(v)$. The requests correspond to the arcs in G, hence $\mathsf{P} = \{(s_e^u, s_e^v) : e = (u, v) \in E\}$ and for each $(p, d) \in \mathsf{P}$ the ride time is set to be $r_{p,d} = a_d - b_p$. Due to the specific numbering of vertices and sizes of windows, the stop p always precedes the stop d and $r_{p,d} \geq 0$ for all $(p, d) \in \mathsf{P}$. Setting the penalty coefficients $\alpha_{p,d} = 0$ and $\beta_{p,d} = 1$, the cost of a schedule corresponds to the number of violated ride time constraints. Using Remark 5.1, let $M = \max_{(p,d)\in\mathsf{P}} \sigma_{p,d}^{RT}(a_{2n})$ and set $\alpha_s = 0$ and $\beta_s = mM + 1$ for all $s \in \mathsf{S}$. Note that $M = 1$, and thus a schedule violating a time window constraint has cost at least $m + 1$. Obviously, the transformation from G to the instance I can be done in polynomial time. An example of this transformation from a DAG to an instance of Min PDS with hard time constraints is depicted in Fig. 5.1.

Obviously, Construction 5.1 can be done in polynomial time.

Theorem 5.1

Min PDS is NP-hard, even with hard time window constraints.

Proof. Let $G = (V, E)$ be a connected DAG such that $|V| = n$, $|E| = m$, and let I be the instance of Min PDS obtained through Construction 5.1. We show that G has a directed cut (A, B) of size at least $(m - k)$ if and only if there exists a schedule t of I with cost at most k, for any $k \in \mathbb{N}$.

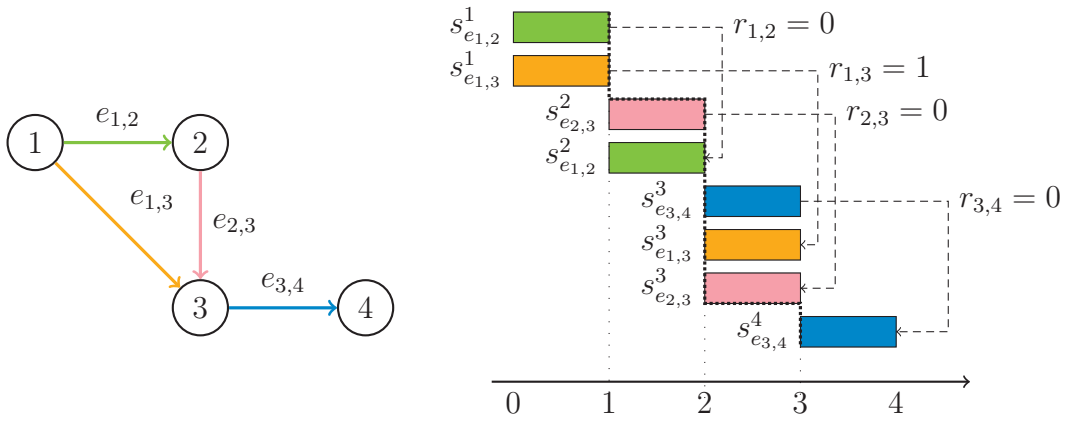


Figure 5.1: A DAG (left) transformed into an instance of PDS (right). Coloured boxes are time windows of length 1. Note there are two stops for every arc of G and the stops are grouped in gadgets for each vertex of G.

Note that a schedule violating a time window constraint has cost $m+1$, and therefore both problems become trivial for $k \geq m+1$. Thus, we suppose that $k \leq m$, and hence no time window can be violated in the schedule.

$\Rightarrow$ Suppose there exists a directed cut (A, B) in G of size at least $(m-k)$. Define a schedule t such that for every vertex $v \in A$ and every stop $s \in \mathsf{S}^v$ we set $\mathsf{t}_s := b_s$ and for all other stops $\mathsf{t}_s := a_s$. Clearly, t is a feasible schedule. Each arc $(u, v) \in E$ corresponds to the unique $(p, d) \in \mathsf{P}$ with $p \in \mathsf{S}^u$ and $d \in \mathsf{S}^v$. If $u \in A$ and $v \in B$, then $\mathsf{t}_d - \mathsf{t}_p = a_d - b_p \leq r_{p,d}$, hence the ride time constraint is respected. As we suppose $s(A, B) \geq m-k$, the previous holds for at least $(m-k)$ requests. With $|\mathsf{P}| = m$, it implies t violates at most k ride time constraints, and hence that t has cost k.

$\Leftarrow$ Now suppose there exists a schedule t for I with cost at most k, that is, violating at most k ride time constraints. For each vertex $v \in V$, let $s[v]$ be the first delivery stop in S^v and if there is no such stop, then $s[v]$ be the last pickup stop in S^v. This allows us to define a partition of V in the following way: for each vertex $v \in V$, if $\mathsf{t}_{s[v]} = b_{s[v]}$ then $v \in A$, otherwise $v \in B$. Choose a $(p, d) \in \mathsf{P}$ and let $u, v \in V$ be such vertices that p is from the gadget S^u and d is from S^v. If t satisfies the ride time constraint of (p, d), then by the definition of I it must hold $\mathsf{t}_p = b_p$ and $\mathsf{t}_d = a_d$. Since p is from the gadget S^u and t is feasible, $\mathsf{t}_{s[v]} \geq b_p$ and necessarily $b_{s[u]} = b_p$, hence $\mathsf{t}_{s[u]} = b_{s[u]}$. Analogously, $\mathsf{t}_{s[v]} = a_{s[v]}$. Therefore, $u \in A$ and $v \in B$. As we suppose that in the schedule t at most k ride time constraints are violated, then at least $|\mathsf{P}| - k = m - k$ are satisfied. Since each satisfied ride time constraint leads to a distinct arc going from A to B, $s(A, B) \geq m-k$. ■

Theorem 5.2

MIN PICKUP-DELIVERY SCHEDULING is APX-hard, even with hard time window constraints.

Proof. We prove that the reduction defined in Theorem 5.1 is in fact an *L-reduction* (see Definition 1.14) from MAXIMUM DICUT to MIN PDS. Let I be an instance of MAXIMUM DICUT on a DAG $G = (V, E)$ with m edges and construct the instance I' of MIN PDS from G as in Theorem 5.1. We know that any optimal solution of MAXIMUM DICUT contains at least $\frac{m}{4}$ edges [3]. Thus, $OPT(I) \geq \frac{m}{4}$, which implies that $OPT(I') = m - OPT(I) \leq 4 \cdot OPT(I) - OPT(I) = 3 \cdot OPT(I)$. Also, $|OPT(I) - (m-k)| = |(m - OPT(I')) - (m-k)| = |OPT(I') - k|$. Since MAXIMUM DICUT is APX-hard even when restricted on DAGs [63], the result follows. ■

5.3.2. Scheduling with hard ride time constraints

Now, we study the variant of MIN PICKUP-DELIVERY SCHEDULING in which the ride time constraints must be respected (hard constraints), while time windows may be violated in return for penalty (soft constraints). We prove it can be solved in linear time, despite the NP-hardness of MIN PICKUP-DELIVERY SCHEDULING with soft ride time constraints, as shown in Section 5.3.1.

As it was mentioned in Section 5.2, we consider a model in which *lateness* is the only possible way to violate a time window restriction. When both ride times and time window constraints are hard, a linear time algorithm was proposed by Firat and Woeginger in [44]. It has also been adapted to handle additional *minimum* ride time constraints in [52]. We show how the same approach can be used to minimise lateness penalties.

Our idea, similar to the one used in [44, 52], is to formulate a difference constraint system (DCS) with variables of the schedule and interpret it as a graph in which the existence of negative weight cycles is equivalent to infeasibility of the DCS. In these papers, it is shown how to apply the single-source shortest path algorithm for interval graphs presented in [7] to test the existence of negative weight cycles in linear time. In case of feasible instances, a solution can be extracted in linear time as well. We point out that this approach will lead to a schedule visiting every stop as late as possible: the scheduled time of each stop is chosen by the length of a shortest path from the start vertex. This path corresponds to a

chain of difference equations and can be seen as the tightest upper bound on the timing value. Since the shortest path lengths are upper bounds this implies that no feasible schedule can visit any of the stops later.

Theorem 5.3

MIN PICKUP-DELIVERY SCHEDULING can be solved in linear time if the ride time constraints are hard.

Proof. For a given instance I of MIN PDS consider the bijection f on S such that $f(x) = 2n - x + 1$. Define the *backward* instance I' with the same sequence of stops and with requests $\mathsf{P}' := \{(f(d), f(p)) : (p, d) \in \mathsf{P}\}$. The time windows are defined by $a'_s := 0$ and $b'_s := a_{2n} - a_{f(s)}$ for all $s \in \mathsf{S}$. The ride time constraints are $r'_{f(d),f(p)} := r_{p,d}$. Notice that a feasible schedule of I' always exists, *e.g.* $(0, \cdots, 0)$ is a feasible solution. However, solving the instance I' with Firat and Woeginger's algorithm [44] yields a feasible schedule t' of I' such that every stop is scheduled as late as possible to be still feasible. We can translate t' to a schedule t of I by setting $\mathsf{t}_s := a_{2n} - \mathsf{t}'_{f(s)}$ for all $s \in \mathsf{S}$. Then t may violate lateness constraints of I, but we will show that it is feasible for I. Firstly, for $s \in \mathsf{S}$, we get $\mathsf{t}_s = a_{2n} - \mathsf{t}'_{f(s)} \geq a_{2n} - b'_{f(s)} = a_{2n} - a_{2n} + a_s = a_s$. When $s \in \mathsf{S}_{<2n}$, we get $\mathsf{t}_s = a_{2n} - \mathsf{t}'_{f(s)} = a_{2n} - \mathsf{t}'_{2n-s+1} \leq a_{2n} - \mathsf{t}'_{2n-s} = a_{2n} - \mathsf{t}'_{f(s+1)} = \mathsf{t}_{s+1}$. Lastly, for $(p, d) \in \mathsf{P}$, we have $\mathsf{t}_d - \mathsf{t}_p = \mathsf{t}'_{f(p)} - \mathsf{t}'_{f(d)} \leq r'_{f(p),f(d)} = r_{p,d}$. Therefore t is a feasible schedule for I. As it was noted earlier, t' visits every stop 'as late as possible' to be still feasible. Since all lateness penalties are non-decreasing, t is an optimal schedule for I. ■

5.4. Bounded maximum ride time

The MIN PICKUP-DELIVERY SCHEDULING problem is NP-hard as it follows from Section 5.3.1. In this section we show that some restrictions on the parameters of the problem improve the complexity of the problem.

We suppose that $\mu \in \mathbb{N}$ is a fixed constant. Let μ-MIN PICKUP-DELIVERY SCHEDULING (μ-MIN PDS) be the restriction of the MIN PICKUP-DELIVERY SCHEDULING problem to the instances with the maximum ride time bounded by μ, that is, $r_{p,d} \leq \mu$ for all $(p, d) \in \mathsf{P}$. In the following we propose a polynomial-time algorithm for μ-MIN PDS.

Given an instance of Min PDS, let $\mathcal{W}$ be the set of all time window bounds for all stops, *i.e.* $\mathcal{W} = \bigcup_{s \in \mathsf{S}} \{a_s, b_s\}$. Firstly, we observe that the visit times of an optimal solution can be chosen from a restricted set of time values. We define the set

$$\widetilde{\mathcal{W}} := \Big(\bigcup_{w \in \mathcal{W}} [w - n\mu, w + n\mu] \Big)_{\geq a_1, \leq a_{2n}} .$$

We say that a schedule t is *defined in* $\widetilde{\mathcal{W}}$ if and only if $\mathsf{t}_s \in \widetilde{\mathcal{W}}$ for all $s \in \mathsf{S}$. Note that $\widetilde{\mathcal{W}} = \mathcal{W}$ in case of 0−Min PDS.

Our goal is to show that it is possible to "shift" the visit times in a schedule without increasing its cost until the schedule is defined in $\widetilde{\mathcal{W}}$. For this purpose we define a so called *closure* of each stop. The closure is a subset of stops for which the timings will be adjusted together.

Definition 5.1

Let I be an instance of Min PDS and t be a feasible schedule of I. For a stop $s \in \mathsf{S}$ the *closure* $\mathcal{G}^{\mathsf{t}}_s$ is the minimal subset of S fulfilling the following:

(a) $s \in \mathcal{G}^{\mathsf{t}}_s$,

(b) if $x \in \mathcal{G}^{\mathsf{t}}_s$ and $y \in \mathsf{S}$ with $\mathsf{t}_x = \mathsf{t}_y$, then $y \in \mathcal{G}^{\mathsf{t}}_s$,

(c) for each $(p, d) \in \mathsf{P}$ such that $\mathsf{t}_d - \mathsf{t}_p \leq r_{p,d}$ and $\{p, d\} \cap \mathcal{G}^{\mathsf{t}}_s \neq \emptyset$ then all stops $x \in \mathsf{S}$ with $p \leq x \leq d$ must be in $\mathcal{G}^{\mathsf{t}}_s$.

Now, we show that we can modify the visit times of all stops included in a closure without violating any new ride time constraints.

Lemma 5.3

Let t be a feasible schedule of an instance I of Min PDS and $\ell \in \mathsf{S}$ be a stop with closure $\mathcal{G}^{\mathsf{t}}_\ell$. For a fixed $\delta \in \{-1, 1\}$ let t' be a schedule such that $\mathsf{t}'_s = \mathsf{t}_s + \delta$ for all $s \in \mathcal{G}^{\mathsf{t}}_s$ and $t'_s = t_s$ for all $s \in \mathsf{S} \setminus \mathcal{G}^{\mathsf{t}}_s$. Then:

(i) t' satisfies all ride time constraints satisfied by t, and

(ii) for all stops $s \in \mathsf{S}_{<2n}$ it holds $\mathsf{t}'_s \leq \mathsf{t}'_{s+1}$.

Proof. (i) As it follows from Definition 5.1(c), for all $(p, d) \in \mathsf{P}$ with $\mathsf{t}_d - \mathsf{t}_p \leq r_{p,d}$, either both p and d are contained in $\mathcal{G}^{\mathsf{t}}_\ell$ or neither of them. If $p, d \notin \mathcal{G}^{\mathsf{t}}_\ell$ then $\mathsf{t}'_d - \mathsf{t}'_p = \mathsf{t}_d - \mathsf{t}_p \leq r_{p,d}$. If $p, d \in \mathcal{G}^{\mathsf{t}}_\ell$ then $\mathsf{t}'_d - \mathsf{t}'_p = \mathsf{t}_d + \delta - \mathsf{t}_p - \delta = \mathsf{t}_d - \mathsf{t}_p \leq r_{p,d}$. Hence in both cases t' respects the ride time constraints.

(ii) Fix $s \in \mathsf{S}_{<2n}$. Since t is feasible it holds $\mathsf{t}_s \leq \mathsf{t}_{s+1}$. By $\delta \in \{-1, 1\}$, $\mathsf{t}'_s > \mathsf{t}'_{s+1}$ would imply $\mathsf{t}_s = \mathsf{t}_{s+1}$ and $\{s, s+1\} \cap \mathcal{G}^{\mathsf{t}}_\ell = 1$, which conflicts with the Definition 5.1 (b). ■

Using Lemma 5.3 we can shift all visit times of stops in a closure until one of the stops reaches a time window border. As it follows from the following lemma, this implies that all stops in the closures are *close* to this border and thus in $\widetilde{\mathcal{W}}$.

Lemma 5.4

Let t be a feasible schedule of an instance I of μ-Min PDS and let $s \in \mathsf{S}$. If $\mathcal{G}^{\mathsf{t}}_s$ contains at least two stops, say x, y, and the stop x precedes y, i.e. $x < y$, then:

(i) if the stop y immediately follows the stop x, i.e. $y = x + 1$, then $t_y \leq \mathsf{t}_x + \mu$, and

(ii) $t_y \leq \mathsf{t}_x + n\mu$.

Proof. (i) Let $y = x + 1$. By the definition of $\mathcal{G}^{\mathsf{t}}_s$, either $\mathsf{t}_x = \mathsf{t}_y$, in which case (i) follows, or there exists $p, d \in \mathcal{G}^{\mathsf{t}}_s$ such that $(p, d) \in \mathsf{P}$, $\mathsf{t}_d - \mathsf{t}_p \leq r_{p,d}$, and $p \leq x < y \leq d$. By the feasibility of t we have $\mathsf{t}_p \leq \mathsf{t}_x$ and $\mathsf{t}_y \leq \mathsf{t}_d$ and $\mathsf{t}_d - \mathsf{t}_p \leq r_{p,d} \leq \mu$. Therefore, $\mathsf{t}_y \leq \mathsf{t}_d \leq \mathsf{t}_p + \mu \leq \mathsf{t}_x + \mu$.
(ii) If all stops in $\mathcal{G}^t_s$ are scheduled at the same time, the claim obviously holds. Otherwise, we consider the partition of $\mathcal{G}^{\mathsf{t}}_s$ into nonempty sets $G_1, \ldots, G_k$ such that: for all $G_i \in \mathcal{G}^{\mathsf{t}}_s$ and all $x, y \in G_i$, it holds $\mathsf{t}_x = \mathsf{t}_y$, and we denote this time by $\mathsf{t}(G_i)$; for all $G_i, G_j \in \mathcal{G}^{\mathsf{t}}_s$, if $i < j$, then $\mathsf{t}(G_i) < \mathsf{t}(G_j)$. Due to the properties of $\mathcal{G}^{\mathsf{t}}_s$, for each $i < k$ there exists $i < j \leq k$ such that there is a $(p, d) \in \mathsf{P}$ with $p \in G_i$, $d \in G_j$, and $\mathsf{t}(G_j) \leq \mathsf{t}(G_i) + r_{p,d} \leq \mathsf{t}(G_i) + \mu$. This leads to the observation, that for all $i < k$ we have $\mathsf{t}(G_{i+1}) \leq \mathsf{t}(G_i) + \mu$. Thus, we get $\mathsf{t}(G_k) \leq \mathsf{t}(G_1) + k\mu$. We note that since for each $i < k$ there is a pickup stop in G_i, it is $k \leq n + 1$ and thus $\mathsf{t}(G_k) \leq \mathsf{t}(G_1) + n\mu$. Obviously $\mathsf{t}_x \geq \mathsf{t}(G_1)$ and $\mathsf{t}_y \leq \mathsf{t}(G_k)$ which completes the proof. ■

Theorem 5.4

For a given instance of μ-Min PDS there is an optimal schedule t defined in $\widetilde{\mathcal{W}}$.

Proof. Assume the statement does not hold, then for every optimal schedule t of I there is a stop $\ell \in \mathsf{S}$ such that $\mathsf{t}_\ell \notin \widetilde{\mathcal{W}}$. Let t^* be an optimal schedule maximising $\ell \in \mathsf{S}$ such that $\mathsf{t}^*_s \in \widetilde{\mathcal{W}}$ for all $s < \ell$ and from all such schedulings t^*_ℓ have the maximum value.

Let $\mathcal{G} := \mathcal{G}^{\mathsf{t}^*}_\ell$ be the closure of the stop ℓ. If there is $g \in \mathcal{G}$ with $\mathsf{t}^*_g \in \mathcal{W}$, then, by Lemma 5.4 (ii), $|\mathsf{t}^*_\ell - \mathsf{t}^*_g| \leq n\mu$, and therefore $\mathsf{t}^*_\ell \in \widetilde{\mathcal{W}}$, which contradicts the choice of ℓ. Thus, for all $g \in \mathcal{G}$, necessarily $\mathsf{t}^*_g \notin \mathcal{W}$ and since t^* is feasible we have $\mathsf{t}^*_g > a_g$.

Now, define a schedule t^- and set $\mathsf{t}^-_s := \mathsf{t}^*_s - 1$ for $s \in \mathcal{G}$ and $\mathsf{t}^-_s := \mathsf{t}^*_s$ otherwise. Note that t^- is feasible since for all $g \in \mathcal{G}$ it holds $\mathsf{t}^*_g > a_g$. Let $\mathcal{L}^*_{\mathsf{t}} := \{s \in \mathsf{S} : \mathsf{t}^*_s > b_s\}$ be the set containing all stops of S in which t^* violates the lateness constraint. Similarly, let $\mathcal{R}^*_{\mathsf{t}} := \{(p,d) \in \mathsf{P} : \mathsf{t}^*_d - \mathsf{t}^*_p > r_{p,d}\}$ be the set of all requests from P for which t^* violates the ride time constraints. According to Lemma 5.3, $\mathcal{R}_{\mathsf{t}^-} \subseteq \mathcal{R}_{\mathsf{t}^*}$ and for each stop s we have $\mathsf{t}^-_s \leq \mathsf{t}^-_{s+1}$. Let $L := \mathcal{L}_{\mathsf{t}^*} \cap \mathcal{G}$, $R_1 := \{(p,d) \in \mathcal{R}_{\mathsf{t}^*} : p \notin \mathcal{G}, d \in \mathcal{G}\}$ and $R_2 := \{(p,d) \in \mathcal{R}_{\mathsf{t}^*} : p \in \mathcal{G}, d \notin \mathcal{G}\}$. Then,

$$c(\mathsf{t}^-) = c(\mathsf{t}^*) + \sum_{x \in R_2} \alpha_x - \sum_{x \in R_1} \alpha_x - \sum_{x \in \mathcal{R}_{\mathsf{t}^*} \setminus \mathcal{R}_{\mathsf{t}^-}} \beta_x - \sum_{s \in L} \alpha_s - \sum_{s \in \mathcal{L}_{\mathsf{t}^*} \setminus \mathcal{L}_{\mathsf{t}^-}} \beta_s$$

By the optimality of t^* and feasibility of t^- we have $c(\mathsf{t}^-) \geq c(\mathsf{t}^*)$, yielding

$$\sum_{x \in R_2} \alpha_x \geq \sum_{s \in L} \alpha_s + \sum_{x \in R_1} \alpha_x \tag{5.3}$$

Now, define the schedule t^+ such that $\mathsf{t}^+_s := \mathsf{t}^*_s + 1$ for each $s \in \mathcal{G}$ and $\mathsf{t}^+_s := \mathsf{t}^*_s$ otherwise. According to Lemma 5.3 (ii), t^+ is feasible. Since $\mathsf{t}^*_g \neq b_g \in \mathcal{W}$ for any $g \in \mathcal{G}$, all lateness constraints satisfied in t^* are satisfied in t^+ as well. Moreover, Lemma 5.3 (i) guarantees that all ride time constraints satisfied in t^* are satisfied in t^+. Therefore, we have

$$c(\mathsf{t}^+) \leq c(\mathsf{t}^*) + \sum_{x \in R_1} \alpha_x + \sum_{s \in L} \alpha_s - \sum_{x \in R_2} \alpha_x \overset{\text{by (5.3)}}{\leq} c(\mathsf{t}^*)\,,$$

which implies that t^+ is also optimal. Furthermore, for all stops $s \in \mathsf{S}$ such that $s < \min \mathcal{G}$ it holds $\mathsf{t}^+_s = \mathsf{t}^*_s \in \widetilde{\mathcal{W}}$. Now, due to the choice of t^* and ℓ, we have $\mathsf{t}^+_\ell \notin \widetilde{\mathcal{W}}$. However, $\mathsf{t}^+_\ell > \mathsf{t}^*_\ell$, which contradicts the choice of t^* regarding the maximum value of t^*_ℓ. ■

Our next goal is to show how we can compute an optimal schedule in polynomial time, using the fact that we know that at least one optimal schedule is defined in $\widetilde{\mathcal{W}}$, as proved in Theorem 5.4.

We define $\mathcal{J}_s := \{(p,d) \in \mathsf{P} : p \leq s \text{ and } s < d\}$ as the set of *loaded requests* after the stop $s \in \mathsf{S}$, and its size $\mathsf{load}(s) := |\mathcal{J}_s|$

Definition 5.2

For a given schedule t of an instance of μ-Min PDS and a stop $\ell \in \mathsf{S}$ we define the *partial cost* $\tilde{c}(\mathsf{t}, \ell)$ *of* t up to the stop $\ell \in \mathsf{S}$ as

$$\tilde{c}(\mathsf{t}, \ell) := \sum_{s \in \mathsf{S}_{\leq \ell}} \sigma_s^L(\mathsf{t}_s) + \sum_{(p,d) \in \mathcal{J}_\ell} \sigma_{p,d}^{RT}(\mathsf{t}_\ell - \mathsf{t}_p) + \sum_{\substack{(p,d) \in \mathsf{P} \\ d \leq \ell}} \sigma_{p,d}^{RT}(\mathsf{t}_d - \mathsf{t}_p)\,.$$

In the following lemma we prove some observations regarding the partial cost function.

Lemma 5.5

For a given schedule t of the instance μ-Min PDS and the stop $\ell \in \mathsf{S}$ the following hold

(i) $\tilde{c}(\mathsf{t}, 1) = \sigma_1^L(\mathsf{t}_1)$;

(ii) $\tilde{c}(\mathsf{t}, \ell+1) = \tilde{c}(\mathsf{t}, \ell) + \sigma_{\ell+1}^L(\mathsf{t}_{\ell+1}) + \sum_{(p,d) \in \mathcal{J}_\ell} f_{p,d}^\ell(\mathsf{t})$
with $f_{p,d}^\ell(\mathsf{t}) := \begin{cases} \alpha_{p,d}(\mathsf{t}_{\ell+1} - \mathsf{t}_\ell), & \text{if } \mathsf{t}_\ell - \mathsf{t}_p > r_{p,d} \\ \sigma_{p,d}^{RT}(\mathsf{t}_{\ell+1} - \mathsf{t}_p), & \text{if } \mathsf{t}_\ell - \mathsf{t}_p \leq r_{p,d} \end{cases}$,

(iii) $\tilde{c}(\mathsf{t}, 2n) = c(\mathsf{t})$,

Proof. (i) and (iii) follow directly from Definition 5.2.
Regarding (ii), we note that for $\ell \in \mathsf{S}_{<2n}$ it is $\mathcal{J}_{\ell+1} = \mathcal{J}_\ell \cup \{(\ell+1, d)\}$ if $(\ell+1)$ is a pickup stop and $(\ell+1, d) \in \mathsf{P}$, or $\mathcal{J}_{\ell+1} = \mathcal{J}_\ell \setminus \{(p, \ell+1)\}$ if $(l+1)$ is a delivery stop, $(p, \ell+1) \in \mathsf{P}$. Therefore,

$$\begin{aligned} \sum_{(p,d) \in \mathcal{J}_{\ell+1}} \sigma_{p,d}^{RT}(\mathsf{t}_{\ell+1} - \mathsf{t}_p) &= \sum_{(p,d) \in \mathcal{J}_\ell} \sigma_{p,d}^{RT}(\mathsf{t}_{\ell+1} - \mathsf{t}_p) + \sigma_{\ell+1,d}^{RT}(\mathsf{t}_{\ell+1} - \mathsf{t}_{\ell+1}) \\ &= \sum_{(p,d) \in \mathcal{J}_\ell} \sigma_{p,d}^{RT}(\mathsf{t}_{\ell+1} - \mathsf{t}_p) \end{aligned}$$

or

$$\sum_{(p,d)\in\mathcal{J}_{\ell+1}} \sigma^{RT}_{p,d}(\mathsf{t}_{\ell+1}-\mathsf{t}_p) = \sum_{(p,d)\in\mathcal{J}_{\ell}} \sigma^{RT}_{p,d}(\mathsf{t}_{\ell+1}-\mathsf{t}_p) - \sigma^{RT}_{p,\ell+1}(\mathsf{t}_{\ell+1}-\mathsf{t}_p)\,.$$

Thus

$$\begin{aligned}\tilde{c}(\mathsf{t},\ell+1) &= \sum_{s\in\mathsf{S}_{\le\ell+1}} \sigma^{L}_{s}(\mathsf{t}_s) + \sum_{(p,d)\in\mathcal{J}_{\ell+1}} \sigma^{RT}_{p,d}(\mathsf{t}_{\ell+1}-\mathsf{t}_p) + \sum_{\substack{(p,d)\in\mathsf{P}\\ d\le\ell+1}} \sigma^{RT}_{p,d}(\mathsf{t}_d-\mathsf{t}_p)\\ &= \sum_{s\in\mathsf{S}_{\le\ell+1}} \sigma^{L}_{s}(\mathsf{t}_s) + \sum_{(p,d)\in\mathcal{J}_{\ell}} \sigma^{RT}_{p,d}(\mathsf{t}_{\ell+1}-\mathsf{t}_p) + \sum_{\substack{(p,d)\in\mathsf{P}\\ d\le\ell}} \sigma^{RT}_{p,d}(\mathsf{t}_d-\mathsf{t}_p)\\ &= \tilde{c}(\mathsf{t},\ell) + \sigma^{L}_{\ell+1}(\mathsf{t}_{\ell+1}) + \sum_{(p,d)\in\mathcal{J}_{\ell}} \left(\sigma^{RT}_{p,d}(\mathsf{t}_{\ell+1}-\mathsf{t}_p) - \sigma^{RT}_{p,d}(\mathsf{t}_{\ell}-\mathsf{t}_p)\right).\end{aligned}$$

For $(p,d)\in\mathcal{J}_\ell$ we have $\sigma^{RT}_{p,d}(\mathsf{t}_\ell-\mathsf{t}_p)=0$ if $\mathsf{t}_\ell-\mathsf{t}_p\le r_{p,d}$, otherwise $\sigma^{RT}_{p,d}(\mathsf{t}_{\ell+1}-\mathsf{t}_p)-\sigma^{RT}_{p,d}(\mathsf{t}_\ell-\mathsf{t}_p)=\alpha_{p,d}(\mathsf{t}_{\ell+1}-\mathsf{t}_p)+\beta_{p,d}-\alpha_{p,d}(\mathsf{t}_\ell-\mathsf{t}_p)-\beta_{p,d}=\alpha_{p,d}(\mathsf{t}_{\ell+1}-\mathsf{t}_\ell)$. Thus

$$\sum_{(p,d)\in\mathcal{J}_{\ell}} \left(\sigma^{RT}_{p,d}(\mathsf{t}_{\ell+1}-\mathsf{t}_p) - \sigma^{RT}_{p,d}(\mathsf{t}_{\ell}-\mathsf{t}_p)\right) = \sum_{(p,d)\in\mathcal{J}_{\ell}} f^{\ell}_{p,d}(\mathsf{t})\,,$$

which concludes the proof. ■

Let I be an instance of μ-Min PDS. For each stop l, $l=1,2,\ldots,2n$ we define so called *l-labels* to capture the structure of 'similar' schedules for I. The labels enable to restrict the number of schedules for I in each step and therefore to use the idea of dynamic programming.

As it follows from Theorem 5.4, we can focus on schedules defined in $\widetilde{\mathcal{W}}$ only. For a schedule t defined in $\widetilde{\mathcal{W}}$ and $\ell\in\mathsf{S}$, the *ℓ-label of t* is defined as

$$Label_\ell(\mathsf{t}) = \big(\mathsf{t}_\ell, s^0, s^1, \ldots, s^\mu, \tilde{c}(\mathsf{t},\ell)\big),$$

where $s^m := \min\{s\in\mathsf{S} \text{ such that } \mathsf{t}_s \ge \mathsf{t}_\ell - m\}$, i.e. s^m is the first stop of the schedule t visited at or after time $(\mathsf{t}_\ell - m)$ for any m, $0\le m\le\mu$.

Note that every schedule has one such label for each $\ell\in\mathsf{S}$, but a label may describe more (different) schedules. We say that a label $L\in\widetilde{\mathcal{W}}\times\mathsf{S}^{\mu+1}\times\mathbb{Q}^+$ is a feasible ℓ-label if there exists a feasible schedule t for I with $Label_\ell(\mathsf{t})=L$.

In the following lemma we prove that in each stop there is a restriction on the number of labels to consider to find an optimal schedule.

Lemma 5.6: Domination rule

Let I be an instance of μ-Min PDS and the stop $\ell \in \mathsf{S}$ be fixed. Let $L^1 = \left(\tau, s^0, s^1, \ldots, s^\mu, \tilde{c}_1\right)$ and $L^2 = \left(\tau, s^0, s^1, \ldots, s^\mu, \tilde{c}_2\right)$ be feasible ℓ-labels with $\tilde{c}_1 \leq \tilde{c}_2$. Then there is a feasible schedule t with $Label_\ell(\mathsf{t}) = L^1$ such that $c(\mathsf{t}) \leq c(\mathsf{t}')$ for any feasible schedule t' with $Label_\ell(\mathsf{t}') = L^2$. We say that the label L_1 *dominates* the label L_2.

Proof. Let t^2 be a feasible schedule with $Label_\ell(\mathsf{t}^2) = L^2$ which minimises $c(\mathsf{t}^2)$. Let t^1 be a feasible schedule with $Label_\ell(\mathsf{t}^1) = L^1$. Let t be the schedule defined as $\mathsf{t}_s := \mathsf{t}^1_s$ for $s \in \mathsf{S}_{<\ell}$ and $\mathsf{t}_s := \mathsf{t}^2_s$ for $s \in \mathsf{S}_{\geq \ell}$. By this definition (and $\mathsf{t}_\ell = \mathsf{t}^2_\ell = \tau = \mathsf{t}^1_\ell$), $Label_\ell(\mathsf{t}) = L^1$.
Now we show inductively that $\tilde{c}(\mathsf{t}, i) \leq \tilde{c}(\mathsf{t}^2, i)$ for all i, $\ell \leq i \leq 2n$. The case $i = \ell$ holds due to $\tilde{c}_1 \leq \tilde{c}_2$. For the induction step we assume $\tilde{c}(\mathsf{t}, i) \leq \tilde{c}(\mathsf{t}^2, i)$ holds for every i, $\ell \leq i < 2n$ and now we show also validity for $i+1$. Using Lemma 5.5 (ii) we have

$$\tilde{c}(\mathsf{t}, i+1) = \tilde{c}(\mathsf{t}, i) + \sigma^L_{i+1}(\mathsf{t}_{i+1}) + \sum_{(p,d)\in\mathcal{J}_i} f^i_{p,d}(\mathsf{t})\,.$$

By the definition of t we have $\mathsf{t}_i = \mathsf{t}^2_i$ and $\mathsf{t}_{i+1} = \mathsf{t}^2_{i+1}$. Clearly $\sigma^L_{i+1}(\mathsf{t}_{i+1}) = \sigma^L_{i+1}(\mathsf{t}^2_{i+1})$. Next, we show $f^i_{p,d}(\mathsf{t}) = f^i_{p,d}(\mathsf{t}^2)$ for all $(p,d) \in \mathcal{J}_i$. Let $(p,d) \in \mathcal{J}_i$.

- If $p \geq s^\mu$ we show that $\mathsf{t}_p = \mathsf{t}^2_p$. For contradiction we suppose $\mathsf{t}_p \neq \mathsf{t}^2_p$ and without loss of generality $\mathsf{t}_p > \mathsf{t}^2_p$. If $m := \mathsf{t}_\ell - \mathsf{t}_p$, then following the schedule t it must hold $s^m \leq p$, but following the schedule t^2 also $s^m > p$, a contradiction. Since t and t^2 schedule such stop p at the same time, $f^i_{p,d}(\mathsf{t}) = f^i_{p,d}(\mathsf{t}^2)$.

- If $p < s^\mu$, it means $\mathsf{t}_p < \mathsf{t}_\ell - \mu \leq \mathsf{t}_i - r_{p,d}$ and this leads to $f^i_{p,d}(\mathsf{t}) = \alpha_{p,d}(\mathsf{t}_{i+1} - \mathsf{t}_i)$. The same arguments ensure $f^i_{p,d}(\mathsf{t}^2) = \alpha_{p,d}(\mathsf{t}^2_{i+1} - \mathsf{t}^2_i)$. Due to $\mathsf{t}_i = \mathsf{t}^2_i$ and $\mathsf{t}_{i+1} = \mathsf{t}^2_{i+1}$ we conclude $f^i_{p,d}(\mathsf{t}) = f^i_{p,d}(\mathsf{t}^2)$.

Together with the induction hypothesis we can see that $\tilde{c}(\mathsf{t}, i+1) \leq \tilde{c}(\mathsf{t}^2, i+1)$ and together with Lemma 5.5 (iii), the case $i = 2n$ concludes the proof. ■

Now, according to Lemma 5.6, we can give an upper bound on the number of non-dominated labels for any fixed stop $l \in S$. There are at most $|\widetilde{\mathcal{W}}|$ possibilities for the first item of the label, hence $O(\mu \cdot n^2)$ if $\mu > 0$ (in case $\mu = 0$ only $O(n)$), and $O(n)$ choices for each of the next $(\mu+1)$ items of the label.

Remark 5.2. For each instance of μ-Min PDS and a stop $\ell \in \mathsf{S}$ the number of non-dominated ℓ-labels is bounded by $O(\mu \cdot n^{\mu+3})$ if $\mu > 0$ and by $O(n^2)$ if $\mu = 0$.

This leads to the following results:

Theorem 5.5

An instance of μ-Min PDS with $\mu > 0$ can be solved in time $O(\mu^2 \cdot n^{\mu} \cdot \text{poly}(n))$.

Proof. Starting with the initial labels $\left(\tau, 1, \ldots, 1, \sigma_1^L(\tau)\right)$ for each $\tau \in \widetilde{\mathcal{W}}_{\geq a_1}$ we have all the labels for the first stop for any feasible schedule defined on $\widetilde{\mathcal{W}}$, by Lemma 5.5 (i). The labels for the stop $\ell \in \mathsf{S}_{>1}$ can be calculated from the labels of the stop $(\ell-1)$. For a non-dominated label $(\tau, s^0, \ldots, s^\mu, \tilde{c})$ of the stop $(\ell-1)$ (there are $O(\mu \cdot n^{\mu+3})$ such labels) do the following: for every possible visit time $\tau' \in \widetilde{\mathcal{W}}_{\geq \tau}$ at the stop ℓ (there are $O(\mu \cdot n^2)$ such possible time visits), generate a new label:

- the first item of the label is τ';
- the s^* items are defined in the following way: $(\tau' - \tau)$ items have the value ℓ (truncate to at most $\mu + 1$ items), if $(\tau' - \tau) < \mu + 1$, then start to add the items $s^0, \ldots, s^\mu$ from the previous $(l-1)$-label until there are $(\mu + 1)$ s^*-items,
- the new cost can be calculated in a linear time from the given label using Lemma 5.5 (ii).

Overall each new label is generated in time $O(n)$. Any label at the stop $2n$ minimising the last item of the label (cost) represents only optimal schedules by Lemma 5.5 (iii). ■

Corollary 5.5.1

An instance of $0-$Min PDS can be solved in time $O(n^5)$.

Proof. The result follows from the proof of Theorem 5.5 considering $\widetilde{\mathcal{W}} = \mathcal{W}$ and the fact that the number of non-dominated labels per stop is bounded by $O(n^2)$. ■

We consider another specific case, when the goal is to minimise the sum of the lateness penalties and the sum of the ride times. This implies that $\mu = 0$. Since driving times are excluded from instances of our model (see Section 5.2.1), all ride times can be seen as excess ride times (excess ride times are defined as the actual ride time minus the driving time). The problem can be solved with the algorithm of Dumas et al. [39] in linear time, as proved below. The ride times can also be minimised in a weighted manner, using $\alpha_{p,d} \geq 0$ for $(p,d) \in \mathsf{P}$. The waiting time before a stop $s \in \mathsf{S}_{>1}$ is then simply weighted by $\sum_{(p,d)\in\mathcal{J}_{s-1}} \alpha_{p,d}$.

Proposition 5.1

Minimising lateness penalties and sum of the ride times can be done in linear time.

Proof. Let I be an instance of Min PDS. The cost of a schedule t of I minimising the sum of the lateness penalties and the sum of the ride times is

$$c(\mathsf{t}) = \sum_{s\in\mathsf{S}} \sigma_s^L(\mathsf{t}_s) + \sum_{(p,d)\in\mathsf{P}} (t_d - t_p)\,. \tag{5.4}$$

There is no dependence with the maximum ride time, so we can set $\sigma_{p,d}^{RT}(l) = l$, for all $(p,d) \in \mathsf{P}$, thus Eqs. (5.1) and (5.4) are equivalent. Notice that this is a special case of μ-Min PDS where $\mu = r_{p,d} = 0$, $\alpha_{p,d} = 1$ and $\beta_{p,d} = 0$ for all request $(p,d) \in \mathsf{P}$. We point out that, since the maximum ride times are zero, the cost of a waiting time is directly dependent on the number of requests affected by it, which is exactly the load of the vehicle. Then for a schedule t, we have

$$\sum_{(p,d)\in\mathsf{P}} (\mathsf{t}_d - \mathsf{t}_p) = \sum_{s\in\mathsf{S}<2n} \mathsf{load}(s) \cdot (\mathsf{t}_{s+1} - \mathsf{t}_s)\,. \tag{5.5}$$

This new formulation can be seen as penalised waiting times, hence the problem can be solved with the algorithm of Dumas, Soumis and Desrosiers [39] in linear time. ■

5.5. First pickups then deliveries

In this section we study a class of polynomial-time solvable instance of Min Pickup-Delivery Scheduling. We introduce *First Pickup Then Deliveries* (*FPTD*) instances in which all the stops $1, \ldots, n$ are pickup stops, and the

stops $n+1, \ldots, 2n$ are delivery stops. We show that MIN PICKUP-DELIVERY SCHEDULING can be solved in polynomial time in the class of *FPTD* instances despite the NP-hardness of the problem (Section 5.3.1).

Firstly, we prove that for each stop we can reduce the set of potential scheduling times to a subset polynomial in size. Each time in this subset is calculated from the time windows and maximum ride time values of the given instance.

Definition 5.3

Let t be a schedule of an instance I of MIN PDS and $X \subseteq \mathsf{S}$. We say that X is *extended* if for all $s, s' \in S$, if $s \in X$ and $\mathsf{t}_s = \mathsf{t}_{s'}$ then $s' \in X$.
If X is extended, then we define $\mathsf{t}(X) := \{\mathsf{t}_s : s \in X\}$ as the set of different visit times for the stops in X by t. We also define two altered schedules t^{X+} and t^{X-} by setting, for all $s \in S$, $\mathsf{t}_s^{X+} = \mathsf{t}_s + 1$ and $\mathsf{t}_s^{X-} = \max\{a_s, \mathsf{t}_s - 1\}$ if $t_s \in \mathsf{t}(X)$, and $\mathsf{t}_s^{X+} = \mathsf{t}_s^{X-} = \mathsf{t}_s$ otherwise. Note that if t is a feasible schedule, then both t^{X+} and t^{X-} are feasible schedules. We write t^{s+} and t^{s-} as shorthands for t^{X+} and t^{X-}, with $X = \{x \in S : \mathsf{t}_x = \mathsf{t}_s\}$.

Lemma 5.7

Let I be an *FPTD* instance, t a feasible schedule for I, and $s \leq n$ a pickup stop with $\mathsf{t}_s < \mathsf{t}_{n+1}$. Then, the schedule t^{s+} does not violate a ride time constraint respected by t.

The previous lemma is straightforward, as no pickups are scheduled earlier and no deliveries are scheduled later in t^{s+} compared to t.

Lemma 5.8

Let I be an instance of MIN PDS, t be a schedule of I, and $X \subseteq \mathsf{S}$ be extended and such that $\mathsf{t}_s > a_s$ for all $s \in X$. If $2c(\mathsf{t}) < c(\mathsf{t}^{X-}) + c(\mathsf{t}^{X+})$ then t^{X-} or t^{X+} must violate a constraint satisfied by t.

Proof. To prove a contradiction, we suppose that t^{X-} and t^{X+} only violate a subset of the constraints violated in t and that $2c(\mathsf{t}) < c(\mathsf{t}^{X-}) + c(\mathsf{t}^{X+})$. We define the following sets:

- $X^L := \{x \in X : \mathsf{t}_x > b_x\}$,
- $X^P := \{(p,d) \in \mathsf{P} : p \in X, d \notin X, \mathsf{t}_d - \mathsf{t}_p > r_{p,d}\}$,
- $X^D := \{(p,d) \in \mathsf{P} : d \in X, p \notin X, \mathsf{t}_d - \mathsf{t}_p > r_{p,d}\}$.

Let

$$\Delta := \sum_{x \in X^L} \alpha_x^L - \sum_{(p,d) \in X^P} \alpha_{p,d}^{RT} + \sum_{(p,d) \in X^D} \alpha_{p,d}^{RT}.$$

Since we suppose that t^{X-} and t^{X+} only violate subsets of constraints compared to t, we have $c(\mathsf{t}^{X-}) \leq c(\mathsf{t}) - \Delta$ (note that this makes use of $\mathsf{t}_s > a_s$ for all $s \in X$ since otherwise $\mathsf{t}_s^{X-} = \mathsf{t}_s$) and $c(\mathsf{t}^{X+}) \leq c(\mathsf{t}) + \Delta$. By summing these inequalities, we obtain $2c(\mathsf{t}) \geq c(\mathsf{t}^{X-}) + c(\mathsf{t}^{X+})$, a contradiction. ■

Lemma 5.9

Let I be an *FPTD* instance with $2n$ stops. Then there exists an optimal schedule t of I such that for each $s \in \mathsf{S}$:

- if $\mathsf{t}_s < \mathsf{t}_n$, then $\mathsf{t}_s \in \mathcal{B}^s(t_n)$ such that
  $\mathcal{B}^s(t_n) := \Big(\{b_{s'} : s' \in \mathsf{S},\ s \leq s' \leq n\}_{<t_n} \cup \bigcup_{(p,d)\in\mathsf{P},\ p\leq s} \{\max\{\mathsf{t}_n, a_d\} - r_{p,d}\} \Big)_{\geq a_s}$;
- if $\mathsf{t}_s = \mathsf{t}_n$, then
  $\mathsf{t}_s \in \mathcal{C} := \{a_n\} \cup \bigcup_{(p,d)\in\mathsf{P}} \Big\{ \{b_p, a_d - r_{p,d}\} \cup \bigcup_{(p',d')\in\mathsf{P}} \{b_p + r_{p',d'}, a_d - r_{p,d} + r_{p',d'}\} \Big\}$;
- if $\mathsf{t}_s > \mathsf{t}_n$, then $\mathsf{t}_s = a_s$.

Proof. We consider an optimal schedule t of an *FPTD* instance I. Among all optimal schedules of I, t is chosen

(a) to minimise $\sum_{(p,d)\in\mathsf{P}} \mathsf{t}_d$, the sum of the delivery times, and such that

(b) for all $s \in \mathsf{S}_{\leq n}$ with $\mathsf{t}_s < \mathsf{t}_{n+1}$, any feasible schedule t' with $\mathsf{t}'_s > \mathsf{t}_s$ has $c(t') > c(t)$.

In the following we show that the schedule t has the requested properties. Due to assumptions about our model discussed in Section 5.2.1, we obviously have $\mathsf{t}_{2n} = a_{2n}$ and following (a) the schedule must use 'as soon as possible delivery' strategy, *i.e.* $\mathsf{t}_s = \max\{\mathsf{t}_n, a_s\}$, for every $s \in \mathsf{S}_{>n}$.
Let $s \in \mathsf{S}$. Notice that both t^{s-} and t^{s+} are feasible schedules. Since t is optimal, obviously $c(\mathsf{t}^{s-}) \geq c(\mathsf{t})$. We discuss separately the different options how t_s, t_n, and t_{n+1} can be related and make a conclusion in each case.
Firstly, we assume $\mathsf{t}_s < \mathsf{t}_n$, thus $\mathsf{t}_s < \mathsf{t}_n \leq \mathsf{t}_{n+1}$. Consider the following:

- if t^{s+} violates a constraint satisfied in t, then according to Lemma 5.7, it must be a time window constraint and thus there is a stop $s' \in \mathsf{S}_{\geq s}$ with $b_{s'} = \mathsf{t}_s$, hence $\mathsf{t}_s \in \{b_{s'} : s' \in \mathsf{S},\ s \leq s' \leq n\}_{<t_n, \geq a_s} \subseteq \mathcal{B}^s(t_n)$;

- if t^{s-} violates a constraint satisfied in t, then it must be a ride time constraint, and there is a stop $p \in \mathsf{S}$, $(p,d) \in \mathsf{P}$, such that $\mathsf{t}_d - r_{p,d} = \mathsf{t}_p = \mathsf{t}_s$, hence $\mathsf{t}_s \in \bigcup_{(p,d)\in\mathsf{P},\ p\leq s} \big\{\max\{\mathsf{t}_n, a_d\} - r_{p,d}\big\}_{\geq a_s} \subseteq \mathcal{B}^s(t_n)$.

Secondly, assume that $\mathsf{t}_s = \mathsf{t}_n = \mathsf{t}_{n+1}$. Let K such that $\{s\} \cup \{p : (p,d) \in \mathsf{P},\ \mathsf{t}_p = \mathsf{t}_d - r_{p,d},\ \mathsf{t}_d = \mathsf{t}_s\} \subseteq K$ and extend it if necessary. If $\mathsf{t}_\ell = a_\ell = \mathsf{t}_n$ for $\ell \in K$, we also have $\mathsf{t}_n = a_n \in \mathcal{C}$. If on the other hand $\mathsf{t}_p = a_p < \mathsf{t}_n$ for some $p \in K$, then p is a pickup. By (b) $c(\mathsf{t}^{p+}) > c(\mathsf{t})$, which means there is a stop $\ell \geq p$ with $\mathsf{t}_\ell = \mathsf{t}_p$ and $\mathsf{t}_\ell \geq b_\ell$. Also $\mathsf{t}_\ell = \mathsf{t}_p = a_p \leq a_\ell \leq b_\ell$, concluding $\mathsf{t}_\ell = b_\ell$. Then $\mathsf{t}_n = b_\ell + r_{p,d} \in \mathcal{C}$. On the other hand, when all $\ell \in K$ have $\mathsf{t}_\ell > a_\ell$, we can make use of Lemma 5.8. By (a) and $(n+1) \in K$, it must hold $c(\mathsf{t}^{K-}) > c(\mathsf{t})$. Further, by optimality of t, $c(\mathsf{t}) \leq c(\mathsf{t}^{K+})$, and according to Lemma 5.8 a constraint satisfied in t is violated by t^{K-} or t^{K+}:

- If t^{K-} violates a constraint satisfied in t, then it must be a maximal ride time constraint of a request $(p,d) \in \mathsf{P}$ with $p \in K, d \notin K$. Therefore, $\mathsf{t}_d > \mathsf{t}_n$ and by (a) it is $\mathsf{t}_d = a_d$ and $\mathsf{t}_p = a_d - r_{p,d}$. Further, by definition of K, either $\mathsf{t}_n = \mathsf{t}_p$ or $\mathsf{t}_n = \mathsf{t}_p + r_{p',d'}$ for some $(p',d') \in \mathsf{P}$. We conclude $\mathsf{t}_s = \mathsf{t}_n \in \mathcal{C}$.

- If t^{K+} violates a constraint satisfied in t we can see that it cannot be a ride time constraint, since for all $(p,d) \in \mathsf{P}$ with $d \in K$ and $\mathsf{t}_d = \mathsf{t}_p + r_{p,d}$, also $p \in K$. Therefore, it is a time window constraint with $\mathsf{t}_s = b_\ell$ for an $\ell \in \mathsf{S}$ or $\mathsf{t}_s = b_p + r_{p,d}$ for a $(p,d) \in \mathsf{P}$, and thus $\mathsf{t}_s \in \mathcal{C}$.

Thirdly, assume $\mathsf{t}_s = \mathsf{t}_n < \mathsf{t}_{n+1}$, then s a is pickup stop. Obviously, $\mathsf{t}_n < a_{n+1}$ due to (a) $\mathsf{t}_d = a_d$ for every delivery $d \in \mathsf{S}_{>n}$. If $t_s = a_\ell$ for some $\ell \leq n$, then by $a_n \geq a_\ell$ and feasibility of t it is $\mathsf{t}_s = a_n \in \mathcal{C}$. Otherwise it follows from Lemma 5.8 that t^{s-} or t^{s+} violates a constraint satisfied in t:

- if t^{s+} violates a constraint satisfied in t, then according to Lemma 5.7, it must be a time window constraint with $\mathsf{t}_n = b_n$, and hence $\mathsf{t}_s \in \mathcal{C}$;

- if t^{s-} violates a constraint satisfied in t, then it must be a ride time constraint with $\mathsf{t}_s = \mathsf{t}_n = a_d - r_{p,d}$, $(p,d) \in \mathsf{P}$, and hence $\mathsf{t}_s \in \mathcal{C}$.

Finally, assume that $\mathsf{t}_s > \mathsf{t}_n$. Then the stop s must be a delivery stop, *i.e.* $s \in \mathsf{S}_{>n}$ and the deliveries being scheduled using 'as soon as possible' strategy according to (a) $\mathsf{t}_s = a_s$. ■

According to Lemma 5.9, there is an optimal schedule t with $\mathsf{t}_n \in \mathcal{C}$, and $|\mathcal{C}|$ is quadratic in the instance size. Moreover, when t_n is fixed, each stop $s \in \mathsf{S}_{<n}$ with $\mathsf{t}_s < \mathsf{t}_n$ belongs to $\mathcal{B}^s(t_n)$, which is linear in size. Obviously, when t_n is fixed, one can schedule all deliveries $d \in \mathsf{S}_{>n}$ at $\mathsf{t}_d := \max\{\mathsf{t}_n, a_d\}$. In the following, we show how an optimal schedule for a fixed t_n can be efficiently calculated.

Let $p \in \mathsf{S}_{\leq n}$ be a pickup stop and $(p, d) \in \mathsf{P}$ its corresponding request. We define the function σ_p^k as the partial cost of scheduling p when $\mathsf{t}_n = k$ such that if p is scheduled at l, then $\sigma_p^k(l) = \sigma_p^L(l) + \sigma_{p,d}^{RT}(\max\{k, a_d\} - l)$. In order to define a recursive equation for calculating the cost of the optimal schedule, we define for each pickup stop $s \in \mathsf{S}_{\leq n}$ the function $T_s^k : \mathbb{N} \to \mathbb{N}$:

$$T_s^k(j) := \begin{cases} \max\left(\mathcal{B}^s(k) \cup \{k\}\right)_{\leq j} & \text{if } \left(\mathcal{B}^s(k) \cup \{k\}\right)_{\leq j} \neq \emptyset, \\ -1 & \text{otherwise.} \end{cases}$$

The call of $T_s^k(j)$ yields the largest time of the set $\mathcal{B}^s(k) \cup \{k\}$ which is smaller than j, or returns -1 if there is no such time. Thus, given $t_n = k$ and a bound j on the visit time for the pickup s, we are able to iterate over the candidate times for s in $\mathcal{B}^s(k) \cup \{k\}$.
Thereby, we can define a recursion table calculating the minimum cost of a schedule t when the value of t_n is fixed.

Lemma 5.10

Let I be an *FPTD* instance, $k \in \mathcal{C}$, t a minimum cost schedule of I such that $\mathsf{t}_n = k$. Then, $c(\mathsf{t}) = C[n, k] + \sum_{d \in \mathsf{S}_{>n}} \sigma_d^L(\max\{k, a_d\})$ with

$$C[i, j] = \begin{cases} \min \left\{ \begin{array}{l} C[i-1, T_{i-1}^k(j)] + \sigma_i^k(j), \\ C[i, T_i^k(j-1)] \end{array} \right\} & \text{if } i \geq 1,\ j \geq a_i, \\ 0 & \text{if } i = 0, \\ \infty & \text{otherwise.} \end{cases} \tag{5.6}$$

Proof. We consider that t schedules the deliveries as soon as possible, *i.e.* $t_d = \max\{k, a_d\}$, for all $d \in \mathsf{S}_{>n}$. We have $c(\mathsf{t}) = \sum_{s \in \mathsf{S}} \sigma_s^L(t_s) + \sum_{(p,d) \in \mathsf{P}} \sigma_{p,d}^{RT}(\mathsf{t}_d - t_p)$. Here, we can write $c(\mathsf{t}) = \sum_{s \in \mathsf{S}} \sigma_s^L(\mathsf{t}_s) +$

$\sum_{(p,d)\in\mathsf{P}} \sigma^{RT}_{(p,d)}(\max\{k, a_d\} - \mathsf{t}_p)$. Thus, by rewriting the sum, we obtain

$$c(\mathsf{t}) = \sum_{(p,d)\in\mathsf{P}} [\sigma^L_p(\mathsf{t}_p) + \sigma^{RT}_{p,d}(\max\{k, a_d\} - \mathsf{t}_p)] + \sum_{d\in\mathsf{S}>n} \sigma^L_d(\max\{k, a_d\}),$$

so we need to prove that

$$C[n, m] = \sum_{(p,d)\in\mathsf{P}} [\sigma^L_p(\mathsf{t}_p) + \sigma^{RT}_{p,d}(\max\{k, a_d\} - \mathsf{t}_p)] = \sum_{p\in\mathsf{S}_{\leq n}} \sigma^k_p(\mathsf{t}_p).$$

It is easy to notice that $C[i, j] = 0 + \sigma^k_1(T^k_1(j_1)) + \cdots + \sigma^k_i(T^k_i(j)) = \sum_{l=1}^{i} \sigma^k_l(T^k_l(j_l))$ with $0 \leq j_1 \leq j_2 \leq \cdots \leq j$. Moreover, we can assume that $T^k_l(j_l) \geq a_l$ for $l \leq i$, otherwise $C[i, j_l] = \infty$, which cannot be a minimum. So the equation only considers times which respect the feasibility of a corresponding schedule. Hence, $C[n, k] = c(\mathsf{t}) - \sum_{d\in\mathsf{S}>n} \sigma^L_d(\max\{k, a_d\})$. ■

Finally, to find the cost of an optimal schedule, we have to compute the value $C[n, k]$ for each $k \in \mathcal{C}$. Algorithm 3 uses dynamic programming to compute this calue in $O(n^4)$ time.

Algorithm 3: Dynamic programming algorithm for *FPTD* instances.

Input: I, an *FPTD* instance.
Output: an optimal schedule of I.

```
1  best := ∞;
2  C* := two-dimensional array;
3  for k ∈ C do                                              // O(n²)
4      Compute C[n, k] with Eq. (5.6) and save the values in C;   // O(n²)
5      score := C[n, k] + Σ_{d∈S>n} σ^L_d(max{k, a_d});
6      if score < best then
7          C* ← C;
8          best ← score;
9  backtrack C* to get t and set t_d := max{t_n, a_d}, ∀d ∈ S>n;   // O(n)
10 return t;
```

Theorem 5.6

An *FPTD* instance I with $2n$ stops can be solved in $O(n^4)$.

Proof. According to Lemma 5.10, Eq. (5.6) calculates the cost of an optimal schedule. The proof shows that each time used by Eq. (5.6) to compute $C[n, k]$ can be used to create a feasible schedule. Then, by backtracking

the computed table C^*, one can obtain an optimal schedule of I in linear time. Compute $C[n,k]$ for a fixed k takes $O(nm)$ for $m = \max_{s\in\mathsf{S}} |\mathcal{B}^s(k)|$. Since $|\mathcal{B}^s(k)| = O(n)$ for all $s \in \mathsf{S}$, independently of k, one can calculate the optimal schedule for a fixed $\mathsf{t}_n = k$ in $O(n^2)$. This step has to be done for each k in $\mathcal{C}$, thus $O(n^2)$ times. Hence the $O(n^4)$ time complexity. ■

5.6. Conclusion

We study a new model of the Dial-A-Ride Problem for the scheduling of fixed sequences with several time constraints typical in the Pickup-and-Delivery scenario. We highlight the key role of soft ride time constraints in the combinatorial complexity of the problem, as they induce the NP-hardness of the problem. Then, we prove that if the ride times are bounded by a constant, we can obtain a polynomial-time algorithm. Finally, we show that instances of the problem with a special structure can be solved efficiently, independently of the timing constraints. We believe that this result can be generalised whenever the number of times a pickup is followed by a delivery in the sequence is bounded.

All our polynomial-time algorithms for restricted instances make use of the fact that earliness at stops can be translated into a linear number of ride time constraints. Therefore, we can take advantage of *increasing time windows opening times* and *as soon as possible deliveries* properties. It may be interesting to exploit these properties in already existing algorithms and heuristics for DARP with ride time constraints.

Future research may consider other types of time constraints and more complex penalty functions in restricted instances.

Chapter 6

Conclusion

Outline

"Play is the highest form of research."

Albert Einstein

Motivated by real life challenges and pure research curiosity, we have studied several problems. In this chapter, we recall these problems, review our different results and propose some directions for future research.

6.1. Deciding if a 2-PDS partition exists

In Chapter 2, we defined the notion of *proportional density*, a new paradigm in graph theory that combines local and global properties of the graph within a same inequation. We focused on the problem of finding a 2-PDS partition in a graph, that is, a partition of the graph into two *proportionally dense subgraphs*. Previous research proposed polynomial-time algorithms to find a 2-PDS partition in several classes of graphs, such as trees and graphs with maximum degree 3. However, the existence of graphs without a 2-PDS partition was unknown and our goal was to settle this question, originally asked in [10]. With a computer based approach, we have been able to find small counterexamples and, from their properties and structural similitudes, we defined an infinite family of graphs without a 2-PDS partition. We noticed that all other graphs we generated up to 10 vertices have at least one connected 2-PDS partition, and hence decided to generate all the connected graphs with 11 vertices to check whether they all admit a 2-PDS partition, and if it is connected. We obtained graphs that have a disconnected 2-PDS partition but not a connected one, and we described an infinite family of such graphs. These results answer our original question and more, but the complexity of deciding if a graph admits a (connected) 2-PDS partition remains unknown. Our infinite families are a good start for finding structural properties of graphs to ensure that they admit, or not, a (connected) 2-PDS partition.

6.2. PDS of maximum size

The notions of *proportional density* and *proportionally dense subgraph* were used again in Chapter 3 to define the Max PDS problem, which consists in finding a PDS of maximum size. The problem is motivated by the concept of 2-PDS partition, though dropping the requirement that all the vertices are in a PDS.

We proved several hardness results for Max PDS, such as its APX-hardness on split graphs and NP-hardness on bipartite graphs. We also show that deciding if a PDS is inclusion-wise maximal is coNP-complete, even on bipartite graphs.

We gave a polynomial-time $\frac{2\cdot(\Delta-1)+1}{\Delta}$-approximation algorithm, with Δ the maximum degee of the graph, proving at the same time that the problem is APX-complete. This algorithm always returns a PDS of size $\lceil \frac{n}{2} \rceil$ or $\lceil \frac{n}{2} \rceil + 1$, which implies that all graphs have a PDS of one of these sizes. However, the returned PDS is not necessarily connected and it would be interesting to look for approximation algorithms returning a connected subgraph.

Then, we considered the problem on Hamiltonian cubic graphs and proved that all Hamiltonian cubic graphs except two have a PDS of size $\lfloor \frac{2n+1}{3} \rfloor$, which is an upper bound on the size of a PDS in a cubic graph. Moreover, such a PDS can be found in linear time if a Hamiltonian cycle is given as an additional input. We conjecture that every cubic graph (apart from our two exceptions) has a PDS of size $\lfloor \frac{2n+1}{3} \rfloor$. We also propose a stronger conjecture stating that for any positive integer d, there exists t such that every connected d-regular graphs with at least t vertices has a PDS of size $\lfloor \frac{(\Delta-1)\cdot n+1}{\Delta} \rfloor$.

6.3. Colourful components problems

In Chapter 4, we considered two graph partitioning problems on vertex-coloured graphs, namely Colourful Components and Colourful Partition. These problems are motivated by comparative genomics, but are also very close to well studied graph problems such as Graph Motif, Multicut and Multi-Multiway Cut.

The first part of our contributions is related to the complexity of the problems on k-caterpillars. In particular, we proved that both problems are NP-complete on binary 4-caterpillars, on ternary 3-caterpillars and on quaternary 2-caterpillars. These results answer an open question raised in [18] regarding the complexity of the problems on trees with maximum degree at most 5. Yet, we were able to give a linear-time algorithm that solves the problem on 1-caterpillars without restricition on the maximum degree and even if the backbone induces a cycle. While the fact that the problem is polynomial-time solvable on 1-caterpillars is not surprising, it is however very interesting to obtain a linear time complexity. Besides, our algorithm is an improvement from the previously known quadratic-time algorithm on paths [36] and applies to a much wider class of graphs. Note that only three open cases remain to obtain a complete dichotomy on k-caterpillars: the complexity on binary 3-caterpillars, binary 2-caterpillars and ternary 2-caterpillars.

We then considered the complexity of Colourful Components in planar graphs with small degree. It was proved in [17] that the problem is NP-complete on 3-coloured graphs with maximum degree 6. Using the same kind of reduction, we were able to prove that the problem remains NP-hard on 5-coloured planar graphs with maximum degree 4 and on 12-coloured planar graphs with maximum degree 3. Our results answer an open question from [18] about the complexity of Colourful Components in ℓ-coloured graphs with maximum degree 5, for some constant ℓ. This hardness result is as good as possible with regard to the maximum degree since the problem becomes polynomial-time solvable on graphs

with maximum degree 2 (disjoint union of paths and cycles). One interesting question is whether the problem remains NP-complete when the number of colours is decreased but the maximum degree is 3 or 4.

6.4. Scheduling with soft time constraints

In Chapter 5, we studied a scheduling problem in the context of Dial-A-Ride problems with fixed route and soft time constraints. An instance is a fixed sequence of stops and a set of requests, associated with time window and ride time constraints. Our model does not allow earliness at stops, and hence each stop is given a time window constraint associated with one penalty function for scheduling the stop late. Each request, *i.e.* an ordered pair of stops representing the pickup and the delivery, is associated with a ride time constraint and a penalty function. The penalty functions are linear and non-decreasing. The goal is to find a schedule of minimum cost, with regard to the constraints.

An interesting first result is that the earliness at stops can be rewritten in terms of ride time constraints. This observation allows us to transform our instances into instances where the earliness at a stop is not allowed, and in turn provides us with useful properties on the instance that we can exploit to design efficient algorithms.

To the best of our knowledge, the complexity of the problem was not known when allowing soft ride time constraints. We showed that the problem is NP-hard even when the time window constraints are hard, that is, when only the ride time constraints can be violated. Then, we proposed several polynomial-time algorithms for restricted types of instances. First, we showed that the problem is polynomial-time solvable whenever the ride time constraints are hard, highlighting their importance in the complexity of the problem. Then, we gave a parameterized algorithm for instances with bounded maximum ride time. We also showed that minimising lateness penalties and sum of the ride times can be done in linear time. Finally, we proved that the underlying structure of some instances can be exploited to obtain efficient algorithms. In this regard, we showed that if all pickups precede all deliveries, then the problem can be solved with a dynamic-programming algorithm. This last result gives hope for a parameterized algorithm for instances in which the number of pickups followed by a delivery is bounded.

Future research may try to develop efficient algorithms for non-linear penalty functions, and eventually consider the approximation of the problem.

Bibliography

[1] A. Adamaszek and A. Popa, 'Algorithmic and hardness results for the colorful components problems', *Algorithmica*, vol. 73, no. 2, pp. 371–388, 2015. DOI: 10.1007/s00453-014-9926-0.

[2] P. Alimonti and V. Kann, 'Some APX-completeness results for cubic graphs', *Theoretical Computer Science*, vol. 237, no. 1, pp. 123–134, 2000. DOI: 10.1016/S0304-3975(98)00158-3.

[3] N. Alon, B. Bollobás, A. Gyárfás, J. Lehel and A. Scott, 'Maximum directed cuts in acyclic digraphs', *Journal of Graph Theory*, vol. 55, no. 1, pp. 1–13, 2007. DOI: 10.1002/jgt.20215.

[4] M. H. Alsuwaiyel, *Algorithms: Design Techniques And Analysis (Revised Edition)*. World Scientific, 2016, vol. 14. DOI: 10.1142/9804.

[5] R. Andersen and K. Chellapilla, 'Finding dense subgraphs with size bounds', in *International Workshop on Algorithms and Models for the Web-Graph*, Springer, 2009, pp. 25–37. DOI: 10.1007/978-3-540-95995-3_3.

[6] Y. Asahiro, R. Hassin and K. Iwama, 'Complexity of finding dense subgraphs', *Discrete Applied Mathematics*, vol. 121, no. 1-3, pp. 15–26, 2002. DOI: 10.1016/S0166-218X(01)00243-8.

[7] M. J. Atallah, D. Z. Chen and D. T. Lee, 'An optimal algorithm for shortest paths on weighted interval and circular-arc graphs, with applications', *Algorithmica*, vol. 14, no. 5, pp. 429–441, 1995. DOI: 10.1007/BF01192049.

[8] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi and V. Kann, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, 1st. Berlin, Heidelberg: Springer-Verlag, 1999, ISBN: 3540654313.

[9] A. Avidor and M. Langberg, 'The multi-multiway cut problem', *Theoretical Computer Science*, vol. 377, no. 1-3, pp. 35–42, 2007. DOI: 10.1016/j.tcs.2007.02.026.

[10] C. Bazgan, J. Chlebíková and T. Pontoizeau, 'Structural and algorithmic properties of 2-community structures', *Algorithmica*, vol. 80, no. 6, pp. 1890–1908, 2018. DOI: 10.1007/s00453-017-0283-7.

[11] C. Bazgan, J. Chlebíková and C. Dallard, 'Graphs without a partition into two proportionally dense subgraphs', *arXiv e-prints*, Submitted to *Information Processing Letters*, 2018. arXiv: 1806.10963 [cs.DM].

[12] C. Bazgan, J. Chlebíková, C. Dallard and T. Pontoizeau, 'Proportionally dense subgraph of maximum size: complexity and approximation', *arXiv e-prints*, Accepted for publication in *Discrete Applied Mathematics*, 2019. DOI: 10.1016/j.dam.2019.07.010. arXiv: 1903.06579 [cs.CC].

[13] C. Bazgan, Z. Tuza and D. Vanderpooten, 'The satisfactory partition problem', *Discrete Applied Mathematics*, vol. 154, no. 8, pp. 1236–1245, 2006. DOI: 10.1016/j.dam.2005.10.014.

[14] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo and T. Pitassi, 'The relative complexity of NP search problems', *Journal of Computer and System Sciences*, vol. 57, no. 1, pp. 3–19, 1998. DOI: 10.1006/jcss.1998.1575.

[15] J. A. Bondy, U. S. R. Murty *et al.*, *Graph theory with applications*. Macmillan London, 1976, vol. 290. DOI: 10.1007/978-1-349-03521-2.

[16] N. Bousquet, J. Daligault and S. Thomassé, 'Multicut is FPT', in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, ACM, 2011, pp. 459–468. DOI: 10.1137/140961808.

[17] S. Bruckner, F. Hüffner, C. Komusiewicz, R. Niedermeier, S. Thiel and J. Uhlmann, 'Partitioning into colorful components by minimum edge deletions', in *Annual Symposium on Combinatorial Pattern Matching*, Springer, 2012, pp. 56–69. DOI: 10.1007/978-3-642-31265-6_5.

[18] L. Bulteau, K. K. Dabrowski, G. Fertin, M. Johnson, D. Paulusma and S. Vialette, 'Finding a Small Number of Colourful Components', in *30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)*, N. Pisanti and S. P. Pissis, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 128, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 20:1–20:14, ISBN: 978-3-95977-103-0. DOI: 10.4230/LIPIcs.CPM.2019.20. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2019/10491.

[19] J. Buresh-Oppenheim and T. Morioka, 'Relativized NP search problems and propositional proof systems', in *Proceedings. 19th IEEE Annual Conference on Computational Complexity, 2004.*, IEEE, 2004, pp. 54–67. DOI: 10.1109/CCC.2004.1313795.

[20] P. Buser, 'Cubic graphs and the first eigenvalue of a riemann surface', *Mathematische Zeitschrift*, vol. 162, no. 1, pp. 87–99, 1978, ISSN: 1432-1823. DOI: 10.1007/BF01437826. [Online]. Available: https://doi.org/10.1007/BF01437826.

[21] P. Buser, 'On the bipartition of graphs', *Discrete applied mathematics*, vol. 9, no. 1, pp. 105–109, 1984. DOI: 10.1016/0166-218X(84)90093-3.

[22] M. Chlebík and J. Chlebíková, 'Complexity of approximating bounded variants of optimization problems', *Theoretical Computer Science*, vol. 354, pp. 320–338, 2006. DOI: 10.1016/j.tcs.2005.11.029.

[23] J. Chlebíková, 'Approximating the maximally balanced connected partition problem in graphs', *Information Processing Letters*, vol. 60, no. 5, pp. 225–230, 1996. DOI: 10.1016/S0020-0190(96)00175-5.

[24] J. Chlebíková and C. Dallard, 'Towards a complexity dichotomy for colourful components problems on k-caterpillars and small-degree planar graphs', in *International Workshop on Combinatorial Algorithms*, Springer, 2019, pp. 136–147. DOI: 10.1007/978-3-030-25005-8_12.

[25] J. Chlebíková, C. Dallard and N. Paulsen, 'Complexity of scheduling for DARP with soft ride times', in *International Conference on Algorithms and Complexity*, Springer, 2019, pp. 149–160. DOI: 10.1007/978-3-030-17402-6_13.

[26] B. Chor, M. Fellows and D. Juedes, 'Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps', in *International Workshop on Graph-Theoretic Concepts in Computer Science*, Springer, 2004, pp. 257–269. DOI: 10.1007/978-3-540-30559-0_22.

[27] H. P. Christos and K. Steiglitz, 'Combinatorial optimization: Algorithms and complexity', *Prentice Hall Inc.*, 1982.

[28] S. Cook, 'The P versus NP problem', *The millennium prize problems*, pp. 87–104, 2000.

[29] S. A. Cook, 'The complexity of theorem-proving procedures', in *Proceedings of the third annual ACM symposium on Theory of computing*, ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047.

[30] J.-F. Cordeau and G. Laporte, 'The dial-a-ride problem (DARP): Variants, modeling issues and algorithms', *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 2, pp. 89–101, 2003. DOI: 10.1007/s10288-002-0009-8.

[31] P. Crescenzi, 'A short guide to approximation preserving reductions', in *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, IEEE, 1997, pp. 262–273. DOI: 10.1109/CCC.1997.612321.

[32] M. Cygan, F. V. Fomin, . Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk and S. Saurabh, *Parameterized algorithms*. Springer, 2015. DOI: 10.1007/978-3-319-21275-3.

[33] M. Cygan, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk and S. Saurabh, 'Minimum bisection is fixed-parameter tractable', *SIAM Journal on Computing*, vol. 48, no. 2, pp. 417–450, 2019. DOI: 10.1137/140988553.

[34] J. Desrosiers, Y. Dumas, M. M. Solomon and F. Soumis, 'Time constrained routing and scheduling', *Handbooks in operations research and management science*, vol. 8, pp. 35–139, 1995.

[35] I. Dinur and S. Safra, 'On the hardness of approximating minimum vertex cover', *Annals of Mathematics*, pp. 439–485, 2005.

[36] R. Dondi and F. Sikora, 'Parameterized complexity and approximation issues for the colorful components problems', *Theoretical Computer Science*, vol. 739, pp. 1–12, 2018, ISSN: 0304-3975. DOI: 10.1016/j.tcs.2018.04.044.

[37] R. G. Downey and M. R. Fellows, 'Fixed-parameter tractability and completeness II: On completeness for W[1]', *Theoretical Computer Science*, vol. 141, no. 1-2, pp. 109–131, 1995. DOI: 10.1016/0304-3975(94)00097-3.

[38] R. G. Downey and M. R. Fellows, *Parameterized complexity*. Springer Science & Business Media, 2012. DOI: 10.1016/S1571-0661(04)00301-9.

[39] Y. Dumas, F. Soumis and J. Desrosiers, 'Optimizing the schedule for a fixed vehicle path with convex inconvenience costs', *Transportation Science*, vol. 24, no. 2, pp. 145–152, 1990.

[40] V. Estivill-Castro and M. Parsa, 'On connected two communities', in *Proceedings of the 36th Australasian Computer Science Conference (ACSC)*, 2013, pp. 23–30.

[41] U. Feige and R. Krauthgamer, 'A polylogarithmic approximation of the minimum bisection', *SIAM Journal on Computing*, vol. 31, no. 4, pp. 1090–1118, 2002. DOI: 10.1137/050640904.

[42] U. Feige, D. Peleg and G. Kortsarz, 'The dense k-subgraph problem', *Algorithmica*, vol. 29, no. 3, pp. 410–421, 2001. DOI: 10.1007/s004530010050.

[43] M. R. Fellows, G. Fertin, D. Hermelin and S. Vialette, 'Upper and lower bounds for finding connected motifs in vertex-colored graphs', *Journal of Computer and System Sciences*, vol. 77, no. 4, pp. 799–811, 2011. DOI: 10.1016/j.jcss.2010.07.003.

[44] M. Firat and G. J. Woeginger, 'Analysis of the dial-a-ride problem of Hunsaker and Savelsbergh', *Operations Research Letters*, vol. 39, no. 1, pp. 32–35, 2011. DOI: 10.1016/j.orl.2010.11.004.

[45] R. W. Floyd, 'Nondeterministic algorithms', *Journal of the ACM (JACM)*, vol. 14, no. 4, pp. 636–644, 1967. DOI: 10.1145/321420.321422.

[46] J. Flum and M. Grohe, *Parameterized complexity theory*. Springer Science & Business Media, 2006.

[47] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[48] M. Gatto, R. Jacob, L. Peeters and A. Schöbel, 'The computational complexity of delay management', in *International Workshop on Graph-Theoretic Concepts in Computer Science*, Springer, 2005, pp. 227–238. DOI: 10.1007/11604686_20.

[49] M. U. Gerber and D. Kobler, 'Algorithmic approach to the satisfactory graph partitioning problem', *European Journal of Operational Research*, vol. 125, no. 2, pp. 283–291, 2000. DOI: 10.1016/S0377-2217(99)00459-2.

[50] M. Gerber and D. Kobler, 'Partitioning a graph to satisfy all vertices', *Swiss Federal Institute of Technology, Lausanne*, 1998.

[51] A. V. Goldberg, 'Finding a maximum density subgraph', EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-84-171, 1984.

[52] T. Gschwind, 'Route feasibility testing and forward time slack for the synchronized pickup and delivery problem', Citeseer, Tech. Rep., 2015. DOI: 10.1007/s00291-018-0544-0.

[53] G. He, J. Liu and C. Zhao, 'Approximation algorithms for some graph partitioning problems', in *Graph Algorithms And Applications 2*, World Scientific, 2004, pp. 21–31. DOI: 10.1142/9789812794741_0002.

[54] S. C. Ho, W. Szeto, Y.-H. Kuo, J. M. Leung, M. Petering and T. W. Tou, 'A survey of dial-a-ride problems: Literature review and recent developments', *Transportation Research Part B: Methodological*, 2018. DOI: 10.1016/j.trb.2018.02.001.

[55] W.-L. Hsu and K.-H. Tsai, 'Linear time algorithms on circular-arc graphs', *Information Processing Letters*, vol. 40, no. 3, pp. 123–129, 1991. DOI: 10.1016/0020-0190(91)90165-E.

[56] L. T. Q. Hung, M. M. Syso, M. L. Weaver and D. B. West, 'Bandwidth and density for block graphs', *Discrete Math.*, vol. 189, no. 1-3, pp. 163–176, 1998, ISSN: 0012-365X.

[57] R. M. Karp, 'Reducibility among combinatorial problems', in *Complexity of computer computations*, Springer, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[58] S. Khot, 'On the power of unique 2-prover 1-round games', in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, ACM, 2002, pp. 767–775. DOI: 10.1109/CCC.2002.1004334.

[59] S. Khot and O. Regev, 'Vertex cover might be hard to approximate to within $2-\varepsilon$', *Journal of Computer and System Sciences*, vol. 74, no. 3, pp. 335–349, 2008.

[60] S. Khuller and B. Saha, 'On finding dense subgraphs', in *36th International Colloquium on Automata, Languages and Programming (ICALP)*, ser. LNCS, Springer-Verlag, vol. 5555, 2009, pp. 597–608. DOI: 10.1007/978-3-642-02927-1_50.

[61] P. Kristiansen, S. M. Hedetniemi and S. T. Hedetniemi, 'Alliances in graphs', *Journal of Combinatorial Mathematics and Combinatorial Computing*, vol. 48, pp. 157–177, 2004.

[62] R. E. Ladner, 'On the structure of polynomial time reducibility', *Journal of the ACM (JACM)*, vol. 22, no. 1, pp. 155–171, 1975. DOI: 10.1145/321864.321877.

[63] M. Lampis, G. Kaouri and V. Mitsou, 'On the algorithmic effectiveness of digraph decompositions and complexity measures', *Discrete Optimization*, vol. 8, no. 1, pp. 129–138, 2011. DOI: 10.1016/j.disopt.2010.03.010.

[64] D. Lichtenstein, 'Planar formulae and their uses', *SIAM Journal on Computing*, vol. 11, no. 2, pp. 329–343, 1982. DOI: 10.1137/0211025.

[65] D. Lokshantov, M. Vatshelle and Y. Villanger, 'Independent set in P_5-free graphs in polynomial time', in *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 2014, pp. 570–581. DOI: 10.1137/1.9781611973402.43.

[66] L. Lovász, 'Graph minor theory', *Bulletin of the American Mathematical Society*, vol. 43, no. 1, pp. 75–86, 2006. DOI: 10.1090/S0273-0979-05-01088-8.

[67] D. Marx and I. Razgon, 'Fixed-parameter tractability of multicut parameterized by the size of the cutset', *SIAM Journal on Computing*, vol. 43, no. 2, pp. 355–388, 2014. DOI: 10.1137/110855247.

[68] D. W. Matula and F. Shahrokhi, 'Sparsest cuts and bottlenecks in graphs', *Discrete Applied Mathematics*, vol. 27, no. 1-2, pp. 113–123, 1990. DOI: 10.1016/0166-218X(90)90133-W.

[69] B. D. McKay and A. Piperno, 'Practical graph isomorphism, ii', *Journal of Symbolic Computation*, vol. 60, pp. 94–112, 2014. DOI: 10.1016/j.jsc.2013.09.003.

[70] N. Megiddo and C. H. Papadimitriou, 'On total functions, existence theorems and computational complexity', *Theoretical Computer Science*, vol. 81, no. 2, pp. 317–324, 1991. DOI: 10.1016/0304-3975(91)90200-L.

[71] N. Misra, 'On the parameterized complexity of colorful components and related problems', in *International Workshop on Combinatorial Algorithms*, Springer, 2018, pp. 237–249. DOI: 10.1007/978-3-319-94667-2_20.

[72] B. Mohar, 'Isoperimetric numbers of graphs', *Journal of combinatorial theory, Series B*, vol. 47, no. 3, pp. 274–291, 1989. DOI: 10.1016/0095-8956(89)90029-4.

[73] M. Olsen, 'A general view on computing communities', *Mathematical Social Sciences*, vol. 66, no. 3, pp. 331–336, 2013. DOI: 10.1016/j.mathsocsci.2013.07.002.

[74] C. H. Papadimitriou, 'On inefficient proofs of existence and complexity classes', in *Annals of Discrete Mathematics*, vol. 51, Elsevier, 1992, pp. 245–250. DOI: 10.1016/S0167-5060(08)70637-X.

[75] C. H. Papadimitriou, 'On the complexity of the parity argument and other inefficient proofs of existence', *Journal of Computer and system Sciences*, vol. 48, no. 3, pp. 498–532, 1994. DOI: 10.1016/S0022-0000(05)80063-7.

[76] C. H. Papadimitriou, *Computational complexity*. John Wiley and Sons Ltd., 2003.

[77] H. Räcke, 'Optimal hierarchical decompositions for congestion minimization in networks', in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, ACM, 2008, pp. 255–264. DOI: 10.1145/1374376.1374415.

[78] R. W. Robinson and N. C. Wormald, 'Almost all cubic graphs are hamiltonian', *Random Structures & Algorithms*, vol. 3, no. 2, pp. 117–125, 1992. DOI: 10.1002/rsa.3240030202.

[79] J. A. Rodríguez-Velázquez, I. G. Yero and J. M. Sigarreta, 'Defensive k-alliances in graphs', *Applied Mathematics Letters*, vol. 22, no. 1, pp. 96–100, 2009. DOI: 10.1016/j.aml.2008.02.012.

[80] K. H. Shafique and R. D. Dutton, 'On satisfactory partitioning of graphs', *Congressus Numerantium*, pp. 183–194, 2002.

[81] J. Tang, Y. Kong, H. Lau and A. W. Ip, 'A note on efficient feasibility testing for dial-a-ride problems', *Operations Research Letters*, vol. 38, no. 5, pp. 405–407, 2010. DOI: 10.1016/j.orl.2010.05.002.

[82] R. E. Tarjan, 'Edge-disjoint spanning trees and depth-first search', *Acta Informatica*, vol. 6, no. 2, pp. 171–185, 1976, ISSN: 1432-0525. DOI: 10.1007/BF00268499.

[83] S. Tippenhauer and W. Muzler, 'On planar 3-SAT and its variants', *Fachbereich Mathematik und Informatik der Freien Universitat Berlin*, 2016.

[84] V. V. Vazirani, *Approximation Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2001, ISBN: 3-540-65367-8. DOI: 10.1007/978-3-662-04565-7.

[85] T. Vidal, T. G. Crainic, M. Gendreau and C. Prins, 'Timing problems and algorithms: Time decisions for sequences of activities', *Networks*, vol. 65, no. 2, pp. 102–128, 2015. DOI: 10.1002/net.21587.

[86] D. Wagner and F. Wagner, 'Between min cut and graph bisection', in *International Symposium on Mathematical Foundations of Computer Science*, Springer, 1993, pp. 744–750. DOI: 10.1007/3-540-57182-5_65.

[87] D. B. West *et al.*, *Introduction to graph theory*. Prentice hall Upper Saddle River, NJ, 1996, vol. 2.

[88] C. Zheng, K. Swenson, E. Lyons and D. Sankoff, 'OMG! Orthologs in multiple genomes–competing graph-theoretical formulations', in *International Workshop on Algorithms in Bioinformatics*, Springer, 2011, pp. 364–375. DOI: 10.1007/978-3-642-23038-7_30.

Certificate of Ethics Review

Project Title: Doctoral thesis

Name: Clément Dallard **User ID:** 832753 **Application Date:** 25-Apr-2019 16:42 **ER Number:** ETHIC-2019-479

You must download your certificate, print a copy and keep it as a record of this review.

It is your responsibility to adhere to the University Ethics Policy and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers and University Health and Safety Policy.

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

You are reminded that as a University of Portsmouth Researcher you are bound by the UKRIO Code of Practice for Research; any breach of this code could lead to action being taken following the University's Procedure for the Investigation of Allegations of Misconduct in Research.

Any changes in the answers to the questions reflecting the design, management or conduct of the research over the course of the project must be notified to the Faculty Ethics Committee. **Any changes that affect the answers given in the questionnaire, not reported to the Faculty Ethics Committee, will invalidate this certificate.**

This ethical review should not be used to infer any comment on the academic merits or methodology of the project. If you have not already done so, you are advised to develop a clear protocol/proposal and ensure that it is independently reviewed by peers or others of appropriate standing. A favourable ethical opinion should not be perceived as permission to proceed with the research; there might be other matters of governance which require further consideration including the agreement of any organisation hosting the research.

(A1) Please briefly describe your project:: **Doctoral thesis, titled "Graph partitions with proportional density and colouring constraints".**
(A2) What faculty do you belong to?: **Technology**
(A3) I am sure that my project requires ethical review by my Faculty Ethics Committee because it includes at least one material ethical issue.: **No**
(A5) Has your project already been externally reviewed?: **No**
(B1) Is the study likely to involve human participants?: **No**
(B2) Are you certain that your project will not involve human subjects or participants?: **Yes**
(C6) Is there any risk to the health & safety of the researcher or members of the research team beyond those that have already been risk assessed?: **No**
(D2) Are there risks of damage to physical and/or ecological environmental features?: **No**
(D4) Are there risks of damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: **No**
(E1) Will the study involve the investigator and/or any participants in activities that could be considered contentious, unacceptable, or illegal, or in any other way harmful to the reputation of the University of Portsmouth?: **No**
(E2) Are there any potentially socially or culturally sensitive issues involved? (e.g. sexual, political, legal/criminal or financial): **No**
(F1) Does the project involve animals in any way?: **No**
(F2) Could the research outputs potentially be harmful to third parties?: **No**
(G1) Please confirm that you have read the University Ethics Policy and have considered the implications for your project.: **Confirmed**
(G2) Please confirm that you have read the UK RIO Code of Practice for Research and will conduct your project in accordance with it.: **Confirmed**
(G3) The University is committed to The Concordat to Support Research Integrity.: **Confirmed**
(G4) Submitting false or incorrect information is a breach of the University Ethics Policy and may be considered as misconduct and be subject to disciplinary action. Please confirm you understand this and agree that the information you have entered is correct.: **Confirmed**

FORM UPR16

Research Ethics Review Checklist

Please include this completed form as an appendix to your thesis (see the Research Degrees Operational Handbook for more information

Postgraduate Research Student (PGRS) Information				**Student ID:**	UP832753
PGRS Name:	Clement Dallard				
Department:	School of Computing	**First Supervisor:**	Janka Chlebikova		
Start Date: (or progression date for Prof Doc students)	**October 2016**				
Study Mode and Route:	Part-time ☐ Full-time ☒	MPhil ☐ PhD ☒	MD ☐ Professional Doctorate ☐		

Title of Thesis:	Graph partitions with proportional density and colouring constraints
Thesis Word Count: (excluding ancillary data)	33503

If you are unsure about any of the following, please contact the local representative on your Faculty Ethics Committee for advice. Please note that it is your responsibility to follow the University's Ethics Policy and any relevant University, academic or professional guidelines in the conduct of your study

Although the Ethics Committee may have given your study a favourable opinion, the final responsibility for the ethical conduct of this work lies with the researcher(s).

UKRIO Finished Research Checklist:
(If you would like to know more about the checklist, please see your Faculty or Departmental Ethics Committee rep or see the online version of the full checklist at: http://www.ukrio.org/what-we-do/code-of-practice-for-research/)

a)	Have all of your research and findings been reported accurately, honestly and within a reasonable time frame?	YES ☒ NO ☐
b)	Have all contributions to knowledge been acknowledged?	YES ☒ NO ☐
c)	Have you complied with all agreements relating to intellectual property, publication and authorship?	YES ☒ NO ☐
d)	Has your research data been retained in a secure and accessible form and will it remain so for the required duration?	YES ☒ NO ☐
e)	Does your research comply with all legal, ethical, and contractual requirements?	YES ☒ NO ☐

Candidate Statement:

I have considered the ethical dimensions of the above named research project, and have successfully obtained the necessary ethical approval(s)

Ethical review number(s) from Faculty Ethics Committee (or from NRES/SCREC):	ETHIC-2019-479

If you have *not* submitted your work for ethical review, and/or you have answered 'No' to one or more of questions a) to e), please explain below why this is so:

Signed *(PGRS)*:	C. DALLARD	**Date:** 17/07/19